

Optimization and (Under/over)fitting

EECS 442 – Jeong Joon Park
Winter 2024, University of Michigan

Administrivia

- HW1 grade will be out today.
- Median: 94/100
- Mean: 90/100
- Doing Great!

Administrivia

- Midterm:
- We will ask mostly conceptual questions.
No rigorous calculations or proofs.
- ~8 questions for 80 minutes.
- Understand important concepts in
lecture/assignments and you'll be fine.
- No practice exam: go through the lectures and
assignments again, understand the concepts.

Regularized Least Squares

Add **regularization** to objective that prefers some solutions:

Before: $\arg \min_w \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \longrightarrow \text{Data Loss}$

After: $\arg \min_w \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$

Data Loss Trade-off Regularization

Want model “smaller”: pay a penalty for \mathbf{w} with big norm

Intuitive Objective: accurate model (low loss) but not too complex (low regularization). λ controls how much of each.

Nearest Neighbors

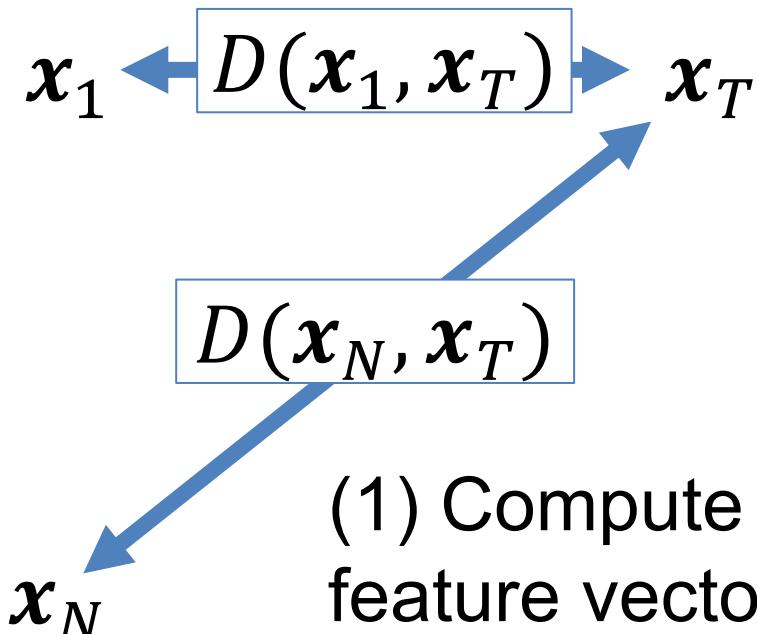
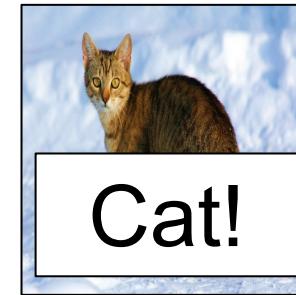
Known Images
Labels



...



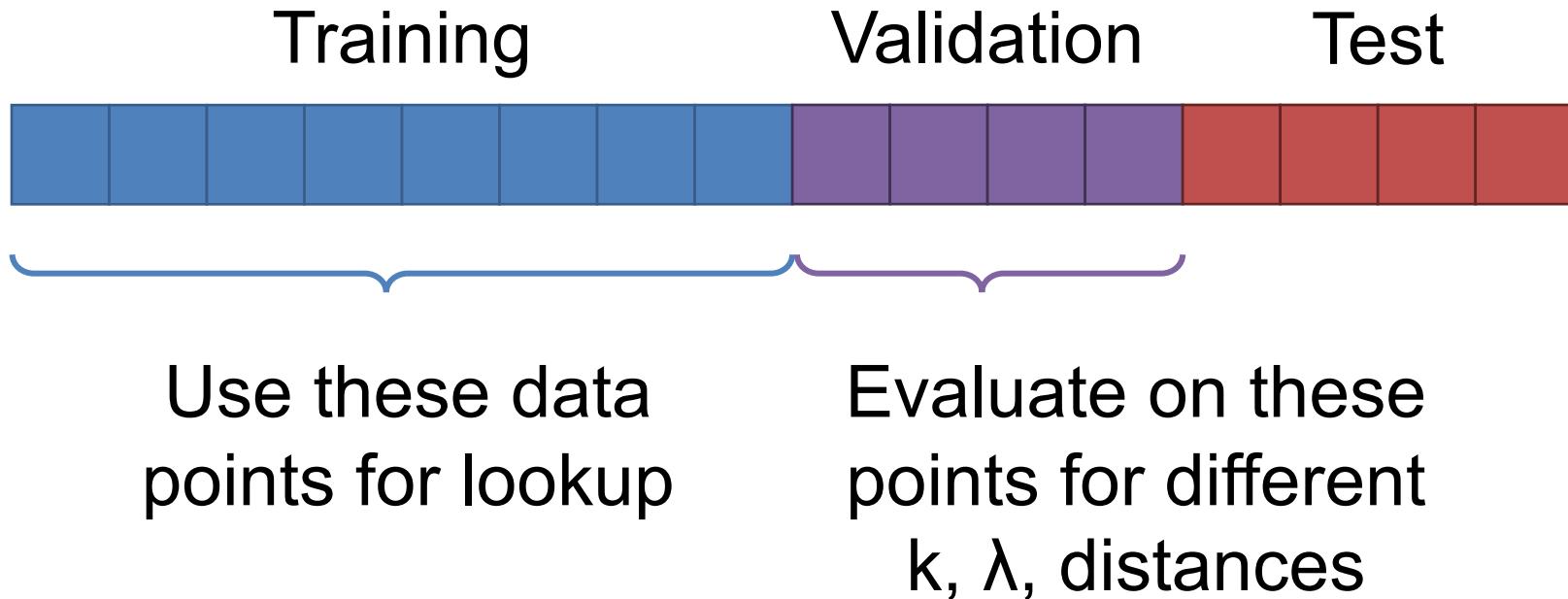
Test
Image



- (1) Compute distance between feature vectors
- (2) find nearest
- (3) use label.

Picking Parameters

What distance? What value for k / λ ?



Linear Models

Example Setup: 3 classes



Model – one weight per class: w_0, w_1, w_2

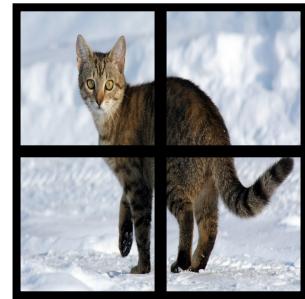
$w_0^T x$ big if cat

Want: $w_1^T x$ big if dog

$w_2^T x$ big if hippo

Stack together: W_{3xF} where x is in \mathbb{R}^F

Linear Models



Cat weight vector

0.2	-0.5	0.1	2.0	1.1
1.5	1.3	2.1	0.0	3.2
0.0	0.3	0.2	-0.3	-1.2

Dog weight vector

0.2	-0.5	0.1	2.0	1.1
1.5	1.3	2.1	0.0	3.2
0.0	0.3	0.2	-0.3	-1.2

Hippo weight vector

W

Weight matrix a collection of scoring functions, one per class

56
231
24
2
1

x



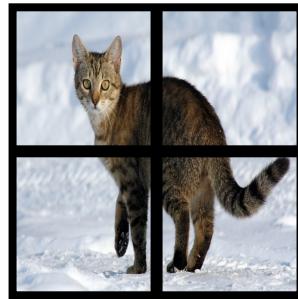
-96.8
437.9
61.95

Cat score
Dog score
Hippo score

Wx

Prediction is vector where jth component is “score” for jth class.

Objective 1: Multiclass SVM



Loss: dog score – cat score

How much higher-scored is
“dog” vs “cat”?

$$(\mathbf{W}\mathbf{x})_2 - (\mathbf{W}\mathbf{x})_1$$

Don't give negative penalties

$$\max(0, (\mathbf{W}\mathbf{x})_2 - (\mathbf{W}\mathbf{x})_1)$$

→ Focus on classification

$(\mathbf{W}\mathbf{x})_1$	-96.8	Cat score
$(\mathbf{W}\mathbf{x})_2$	437.9	Dog score
$(\mathbf{W}\mathbf{x})_3$	61.95	Hippo score

$$\mathbf{W}\mathbf{x}$$

Prediction is vector
where jth component is
“score” for jth class.

Objective 1: Multiclass SVM

Inference (x): $\arg \max_k (\mathbf{W}x)_k$

(Take the class whose weight vector gives the highest score)

Training (\mathbf{x}_i, y_i):

$$\arg \min_{\mathbf{W}} \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n \sum_{j \neq y_i} \max(0, (\mathbf{W}x_i)_j - (\mathbf{W}x_i)_{y_i})$$

Regularization
Over all data points
For every class j that's NOT the correct one (y_i)

Pay no penalty if prediction for class y_i is bigger than j. Otherwise, pay proportional to the score of the wrong class.

Objective 1: Multiclass SVM

Inference (x): $\arg \max_k (\mathbf{W}x)_k$

(Take the class whose weight vector gives the highest score)

Training (\mathbf{x}_i, y_i):

$$\arg \min_{\mathbf{W}} \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n \sum_{j \neq y_i} \max(0, (\mathbf{W}x_i)_j - (\mathbf{W}x_i)_{y_i} + m)$$

Regularization

Over all data points

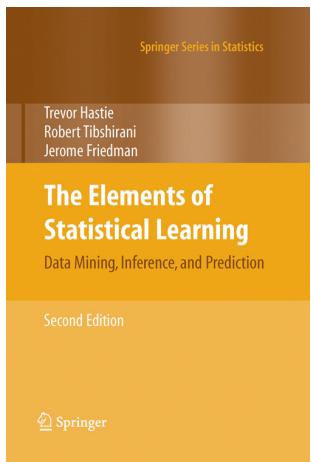
For every class j that's NOT the correct one (y_i)

Pay no penalty if prediction for class y_i is bigger than j by m ("margin"). Otherwise, pay proportional to the score of the wrong class.

Objective 1:

Called: Support Vector Machine

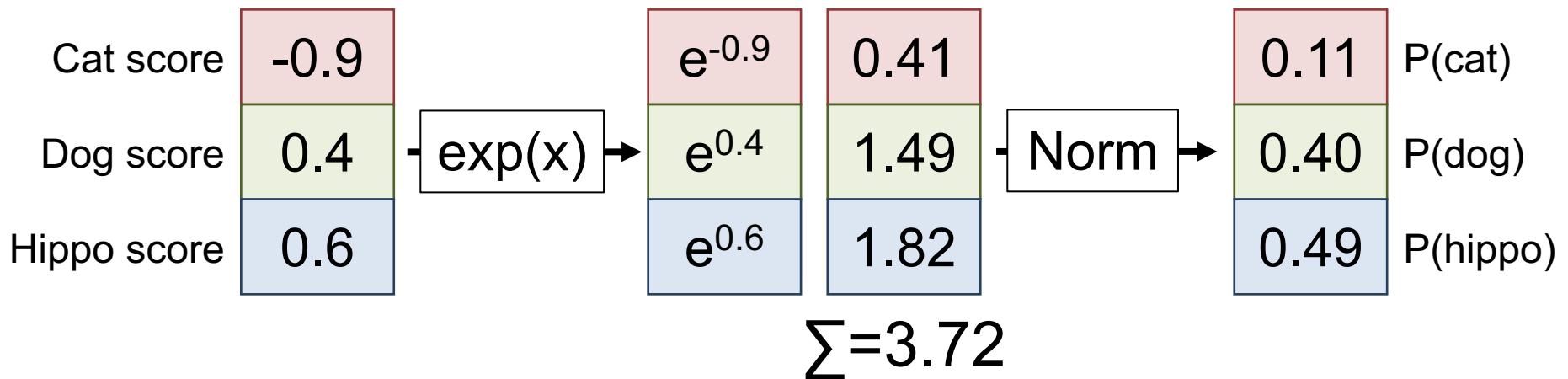
Lots of great theory as to why this is a sensible thing to do. See:



Useful book (Free too!):
The Elements of Statistical Learning
Hastie, Tibshirani, Friedman
<https://web.stanford.edu/~hastie/ElemStatLearn/>

Objective 2: Making Probabilities

Converting Scores to “Probability Distribution”



Generally $P(\text{class } j)$:

$$\frac{\exp((Wx)_j)}{\sum_k \exp((Wx)_k)}$$

Called softmax function

Objective 2: Softmax

Inference (x): $\arg \max_k (Wx)_k$

(Take the class whose weight vector gives the highest score)

$$P(\text{class } j) = \frac{\exp((Wx)_j)}{\sum_k \exp((Wx)_k)}$$

Why can we skip the exp/sum exp thing to make a decision?

Objective 2: Softmax

Inference (x): $\arg \max_k (\mathbf{W}x)_k$

(Take the class whose weight vector gives the highest score)

Training (\mathbf{x}_i, y_i):

$$\arg \min_{\mathbf{W}} \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n -\log \left(\frac{\exp((\mathbf{W}x)_{y_i})}{\sum_k \exp((\mathbf{W}x)_k)} \right)$$

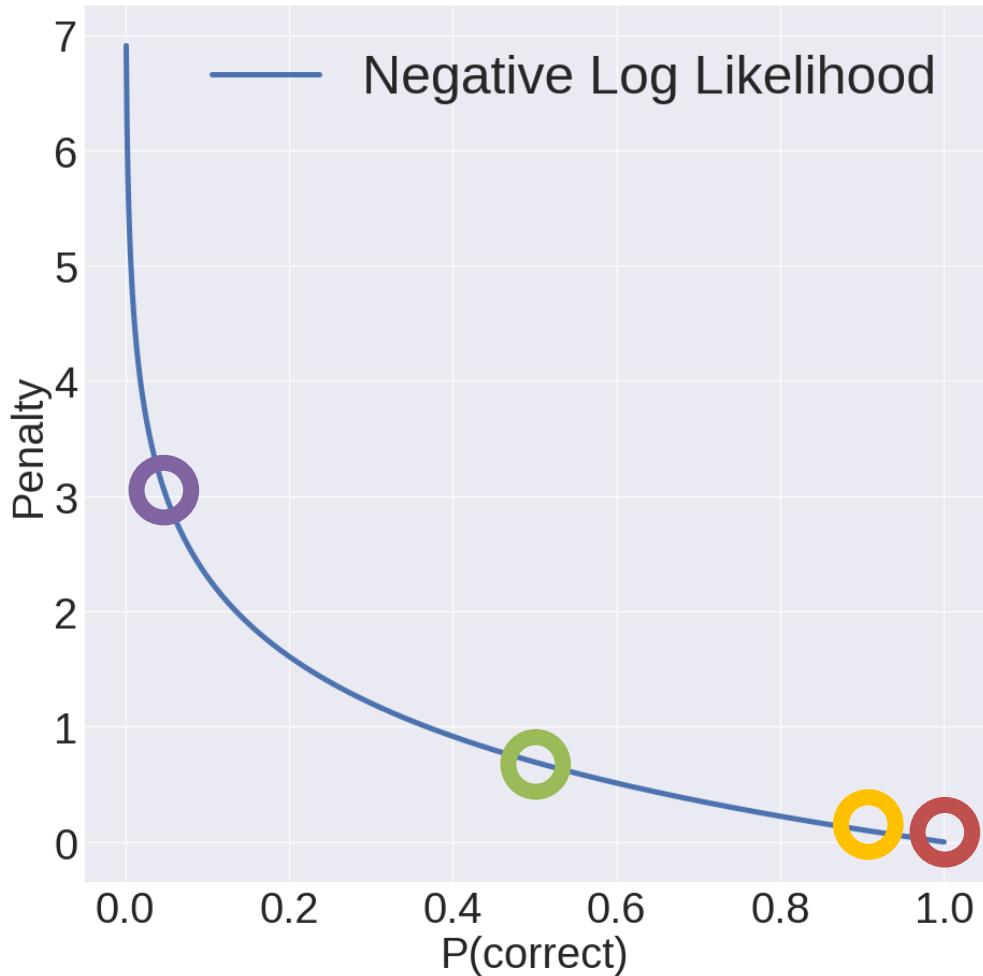
Regularization

Over all data points

P(correct class)

Pay penalty for not making correct class likely.
“Negative log-likelihood”

Negative Log Softmax



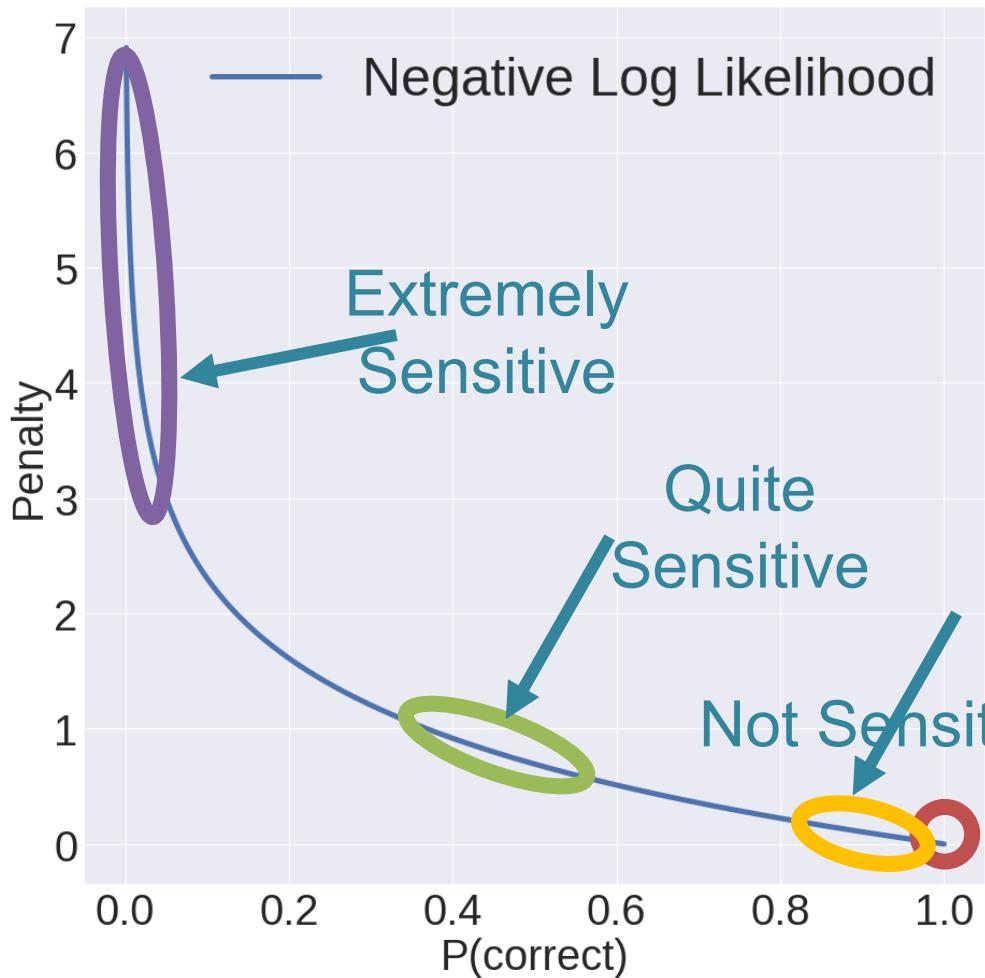
$P(\text{correct}) = 0.05:$
3.0 penalty

$P(\text{correct}) = 0.5:$
0.7 penalty

$P(\text{correct}) = 0.9:$
0.1 penalty

$P(\text{correct}) = 1:$
No penalty!

Negative Log Softmax



$P(\text{correct}) = 0.05$:
3.0 penalty

$P(\text{correct}) = 0.5$:
0.7 penalty

$P(\text{correct}) = 0.9$:
0.1 penalty

$P(\text{correct}) = 1$:
No penalty!

How Do We Optimize Things?

Goal: find the \mathbf{w} minimizing
some loss function L .

$$\arg \min_{\mathbf{w} \in \mathbb{R}^N} L(\mathbf{w})$$

Works for lots of
different L s:

$$L(\mathbf{W}) = \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n -\log\left(\frac{\exp((\mathbf{W}\mathbf{x})_{y_i})}{\sum_k \exp((\mathbf{W}\mathbf{x})_k)}\right)$$

$$L(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

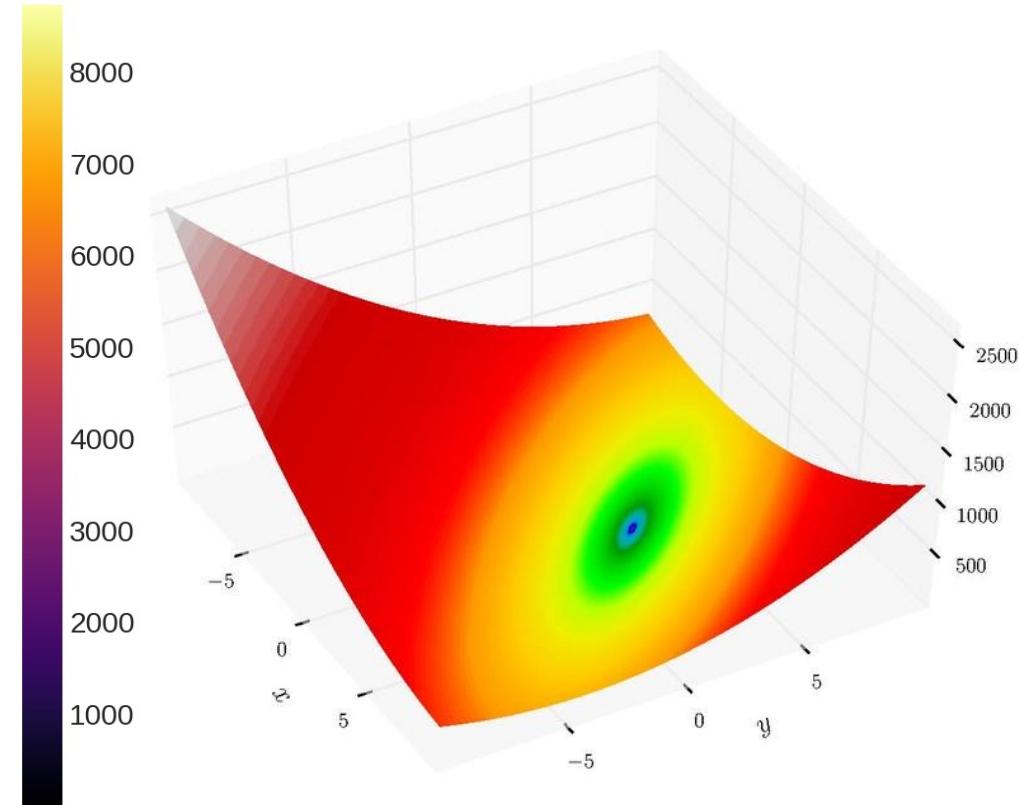
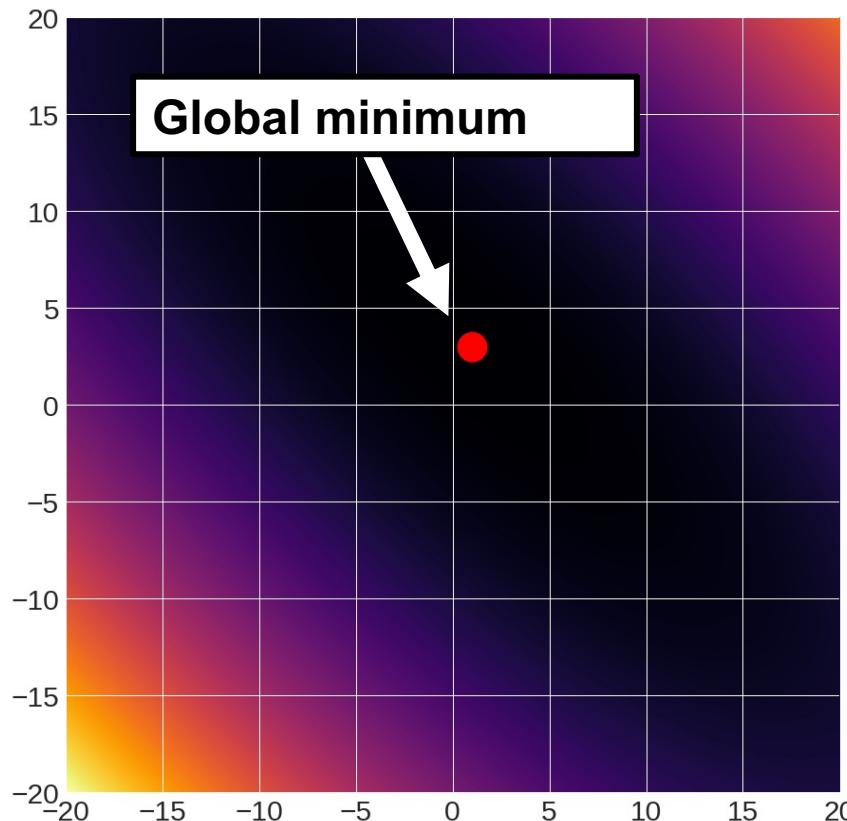
Non-Linear

$$L(\mathbf{w}) = C \|\mathbf{w}\|_2^2 + \sum_{i=1}^n \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

Generally NO Closed-form Solutions!

Sample Function to Optimize

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$$



Right Image Credit: Wikipedia

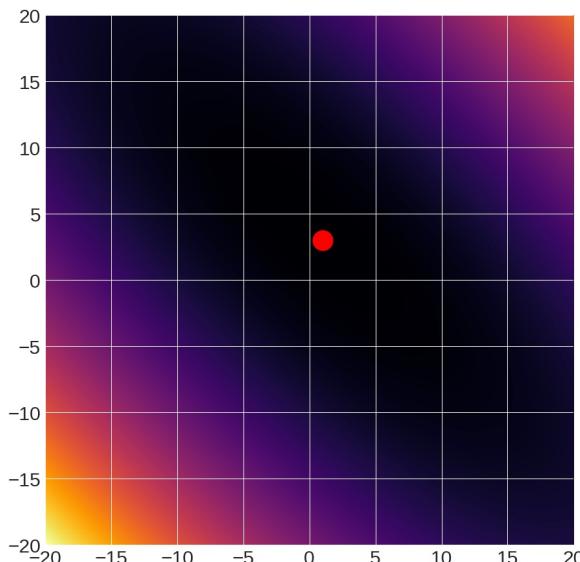
Sample Function to Optimize

- I'll switch back and forth between this 2D function (called the *Booth Function*) and other more-learning-focused functions
- Beauty of optimization is that it's all the same in principle
- But don't draw too many conclusions: 2D space has qualitative differences from 1000D space

See intro of: Dauphin et al. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization* NIPS 2014

<https://ganguli-gang.stanford.edu/pdf/14.SaddlePoint.NIPS.pdf>

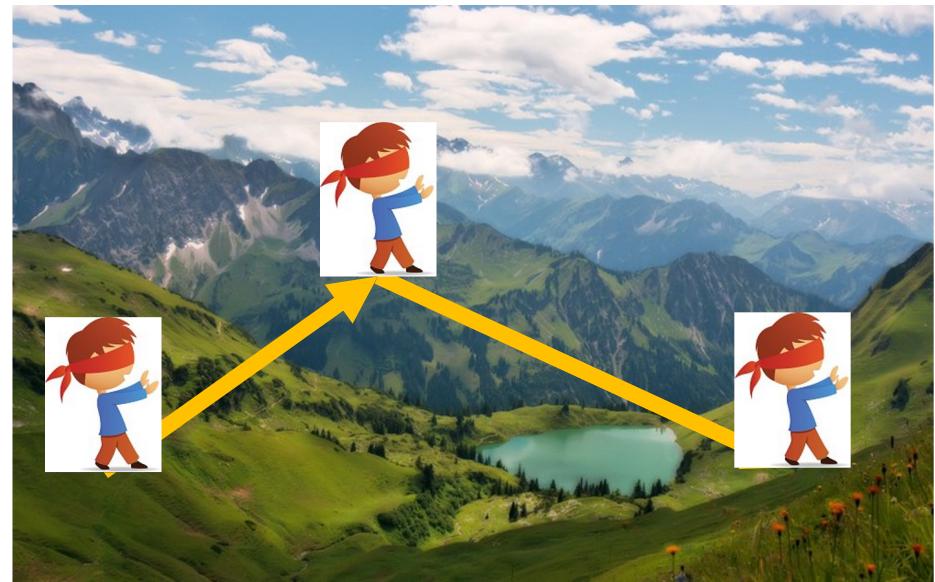
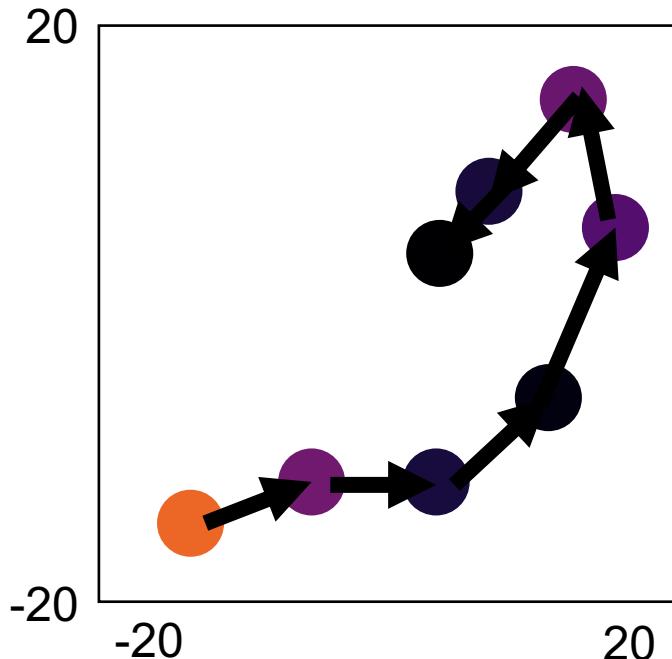
A Caveat



- Each point in the picture is a function evaluation
- Here it takes microseconds – so we can easily see the answer
- Functions we want to optimize may take hours to evaluate

A Caveat

Model in your head: moving around a landscape with a teleportation device

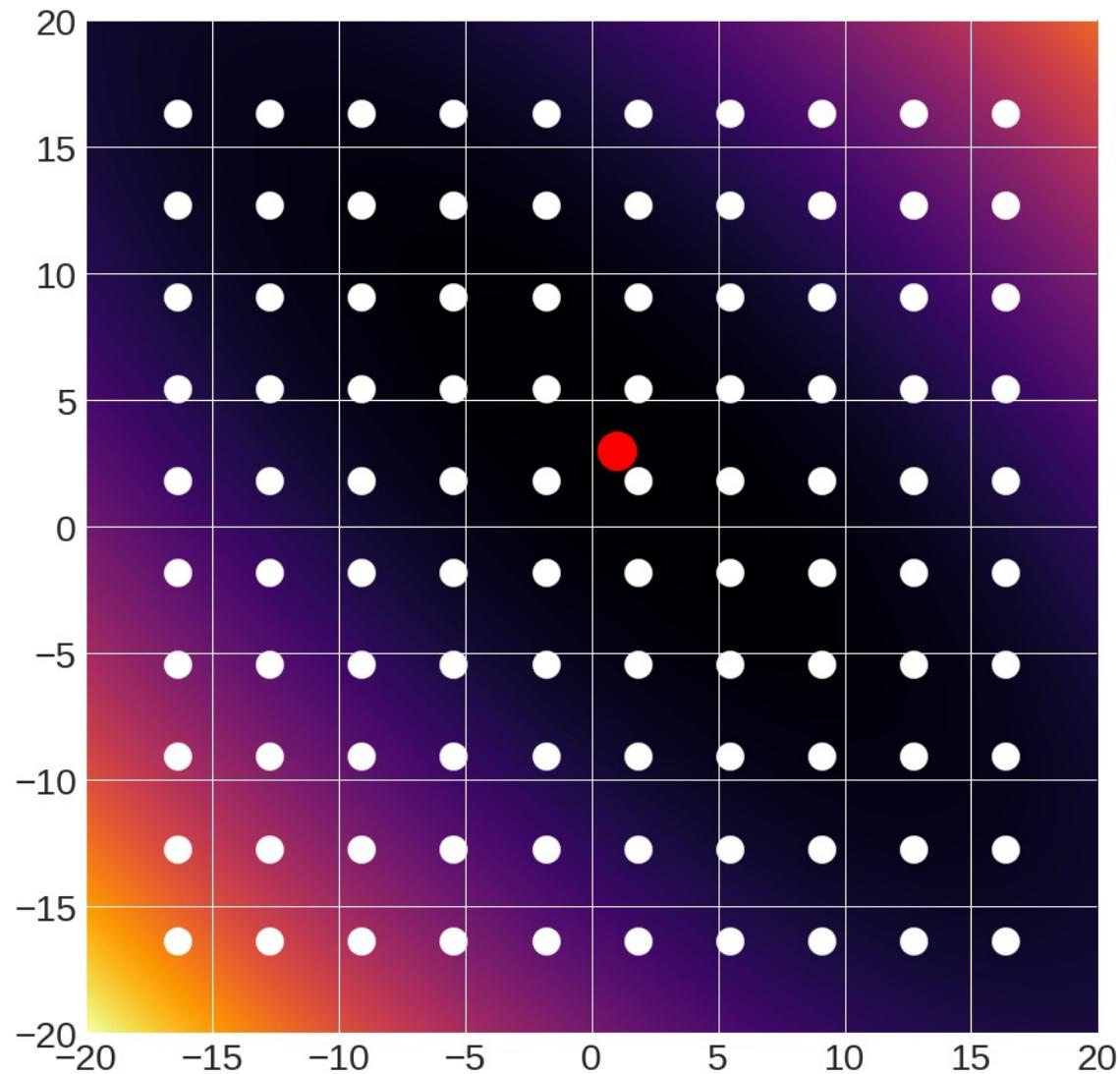


Landscape diagram: Karpathy and Fei-Fei

Option #1A – Grid Search

```
#systematically try things
best, bestScore = None, Inf
for dim1Value in dim1Values:
    ....
    for dimNValue in dimNValues:
        w = [dim1Value, ..., dimNValue]
        if L(w) < bestScore:
            best, bestScore = w, L(w)
return best
```

Option #1A – Grid Search



Option #1A – Grid Search

Pros:

1. Super simple
2. Only requires being able to evaluate model

Cons:

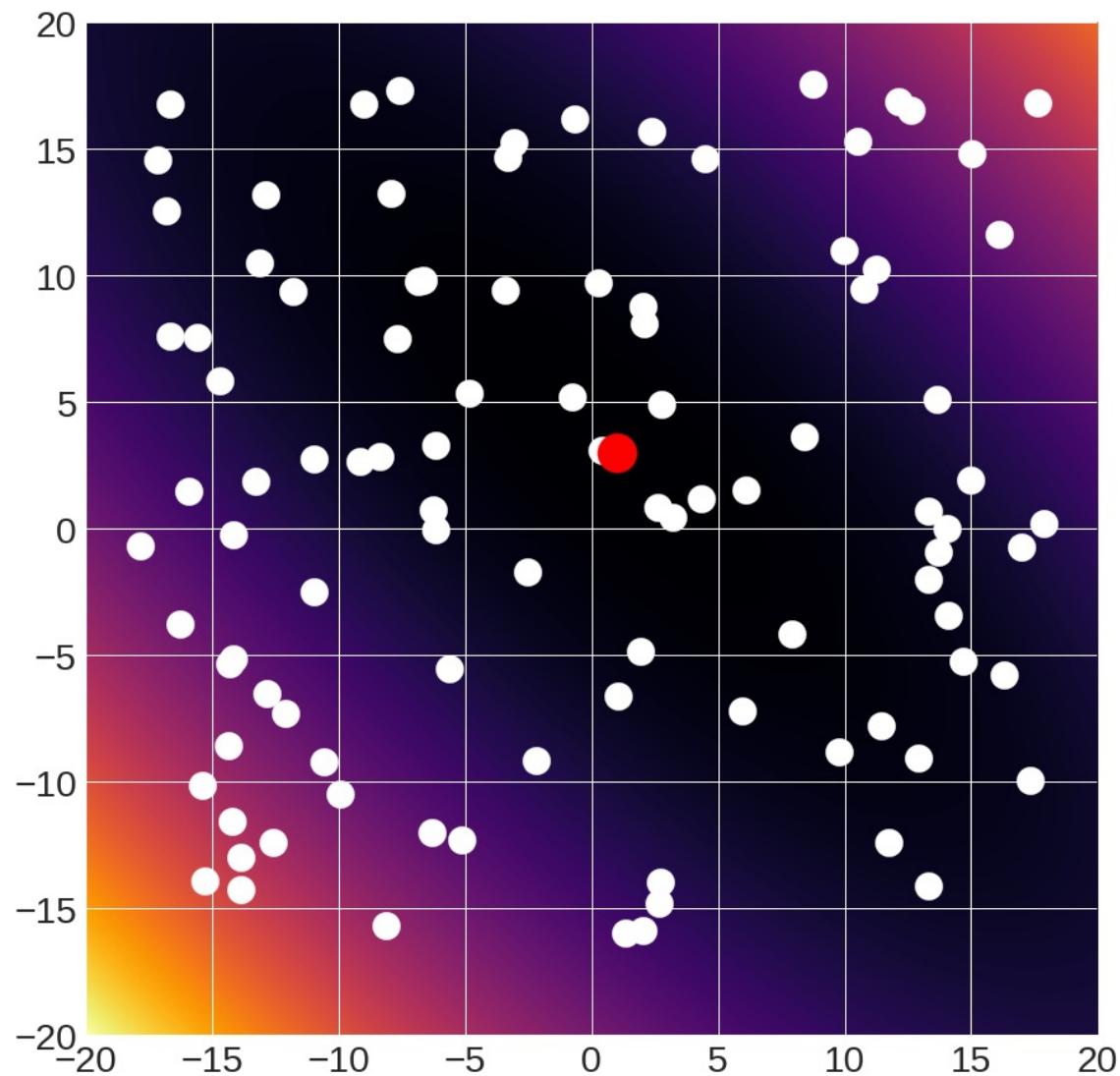
1. Scales horribly to high dimensional spaces

Complexity: $\text{samplesPerDim}^{\text{numberOfDims}}$

Option #1B – Random Search

```
#Do random stuff RANSAC Style
best, bestScore = None, Inf
for iter in range(numIters):
    w = random(N,1) #sample
    score = L(w) #evaluate
    if score < bestScore:
        best, bestScore = w, score
return best
```

Option #1B – Random Search



Option #1B – Random Search

Pros:

1. Super simple
2. Only requires being able to sample model and evaluate it

Cons:

1. Slow – throwing darts at high dimensional dart board
2. Might miss something

$$P(\text{all correct}) = \varepsilon^N$$

Good parameters



All parameters



When Do You Use Options 1A/1B?

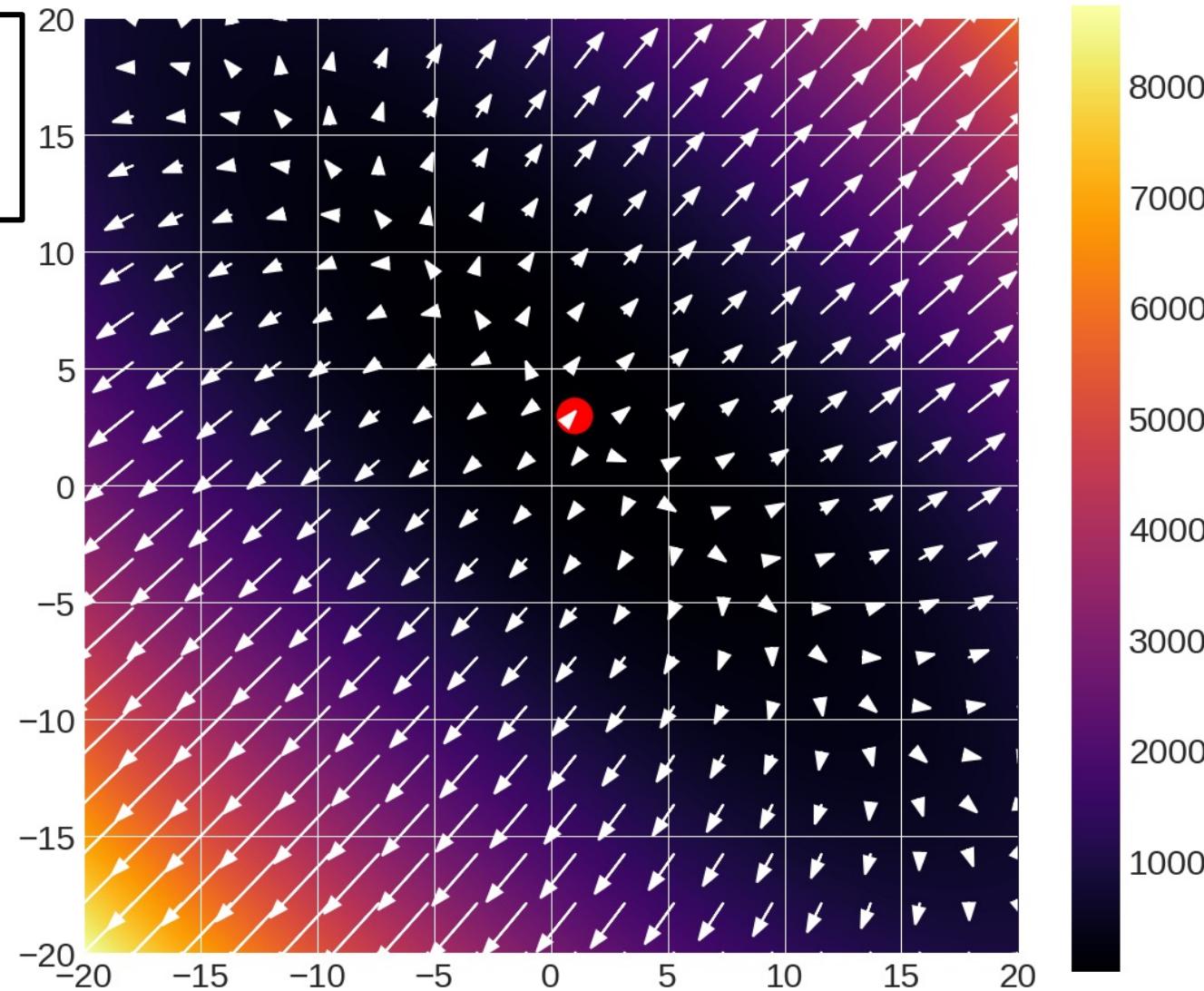
Use these when

- Number of dimensions small, space bounded
- Objective is impossible to analyze (e.g., validation accuracy if one uses a distance function)

Random search is more effective; grid search makes it easy to systematically test something (people love certainty)

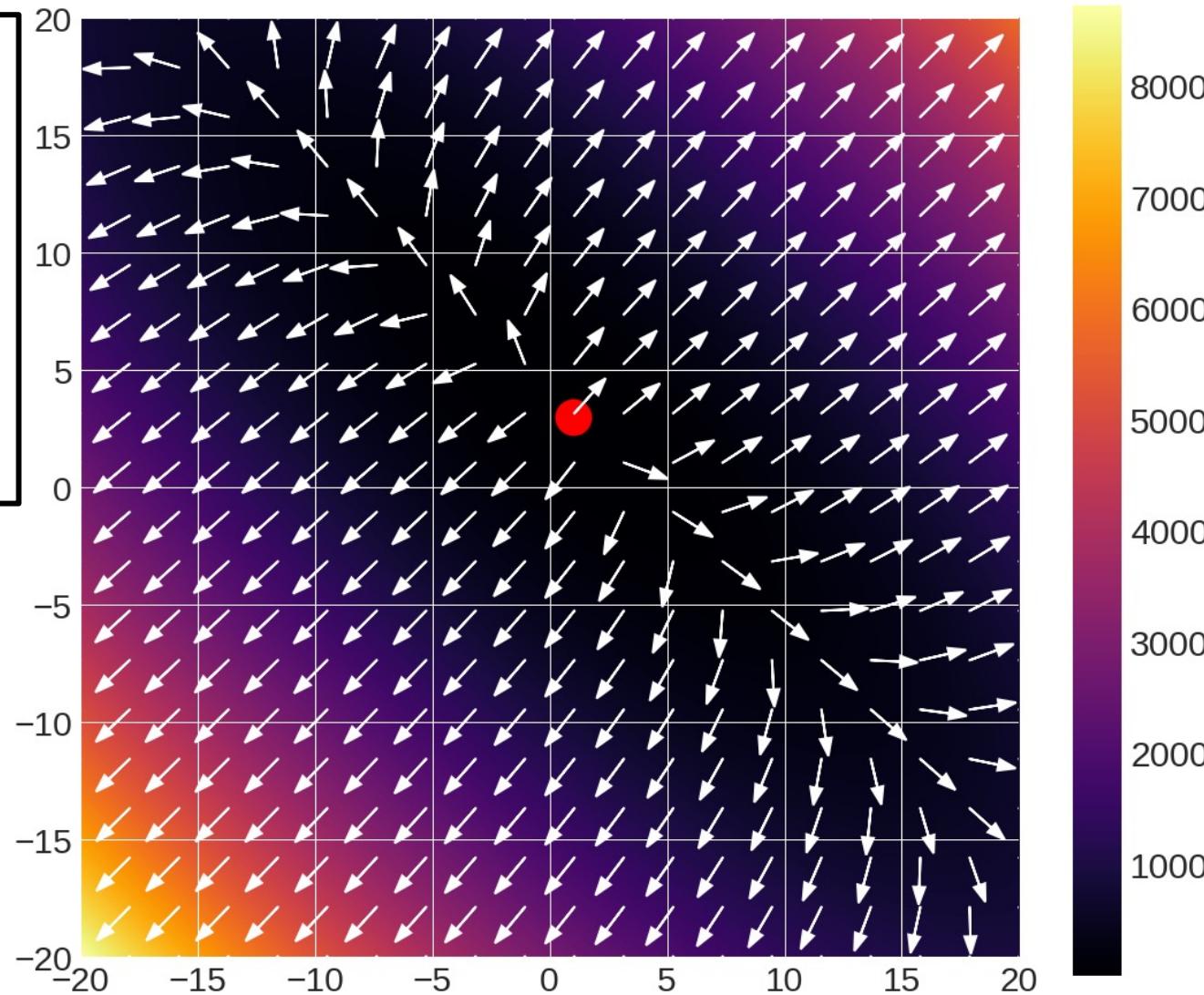
Option 2 – Use The Gradient

Arrows:
gradient



Option 2 – Use The Gradient

Arrows:
gradient direction
(scaled to unit length)



Option 2 – Use The Gradient

Want: $\arg \min_{\mathbf{w}} L(\mathbf{w})$

What's the geometric interpretation of:

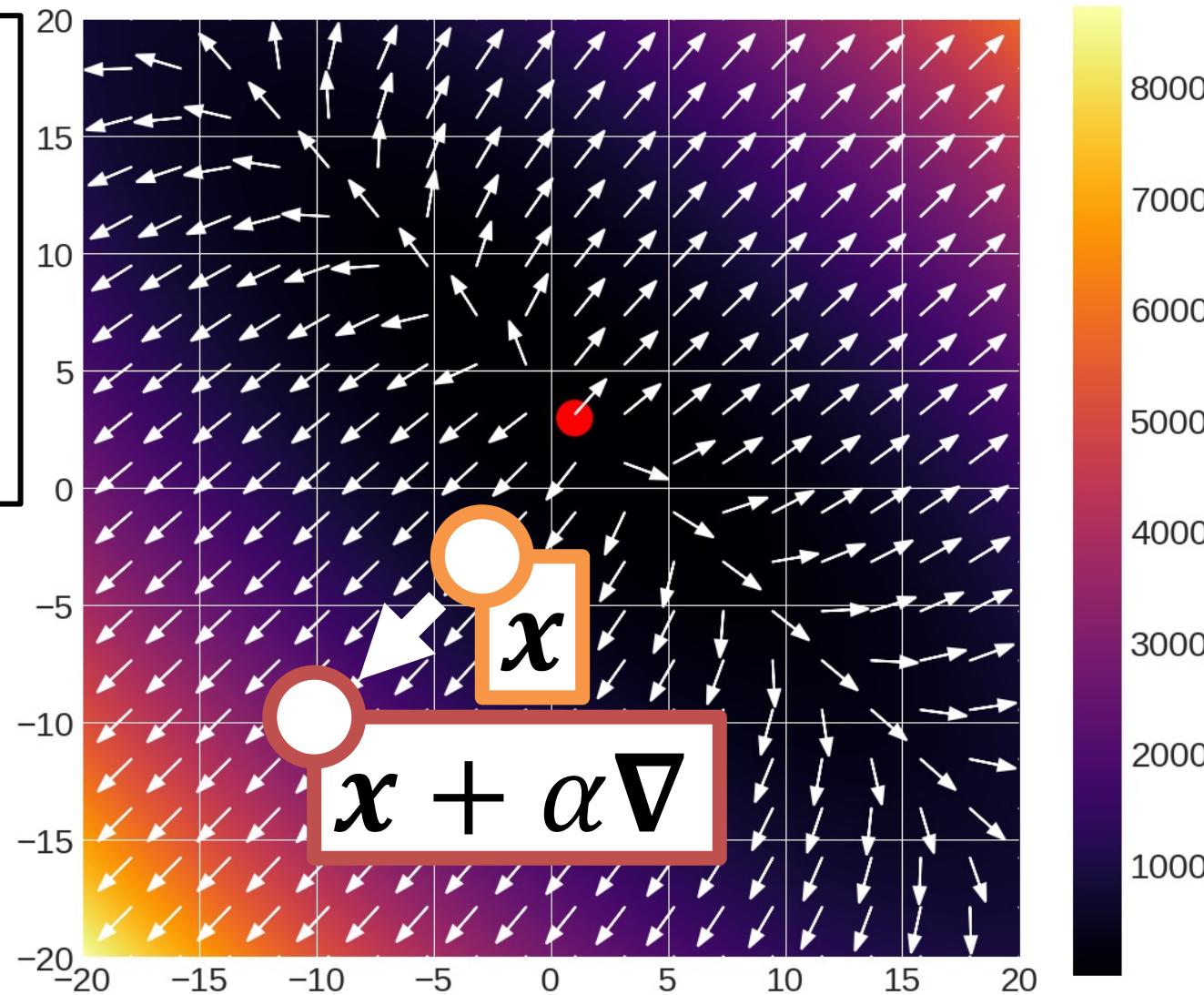
$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \begin{bmatrix} \partial L / \partial x_1 \\ \vdots \\ \partial L / \partial x_N \end{bmatrix}$$

Which is bigger (for small α)?

$$L(\mathbf{w}) \stackrel{\leq ?}{>} L(\mathbf{w} + \alpha \nabla_{\mathbf{w}} L(\mathbf{w}))$$

Option 2 – Use The Gradient

Arrows:
gradient
direction
(scaled to unit
length)



Option 2 – Use The Gradient

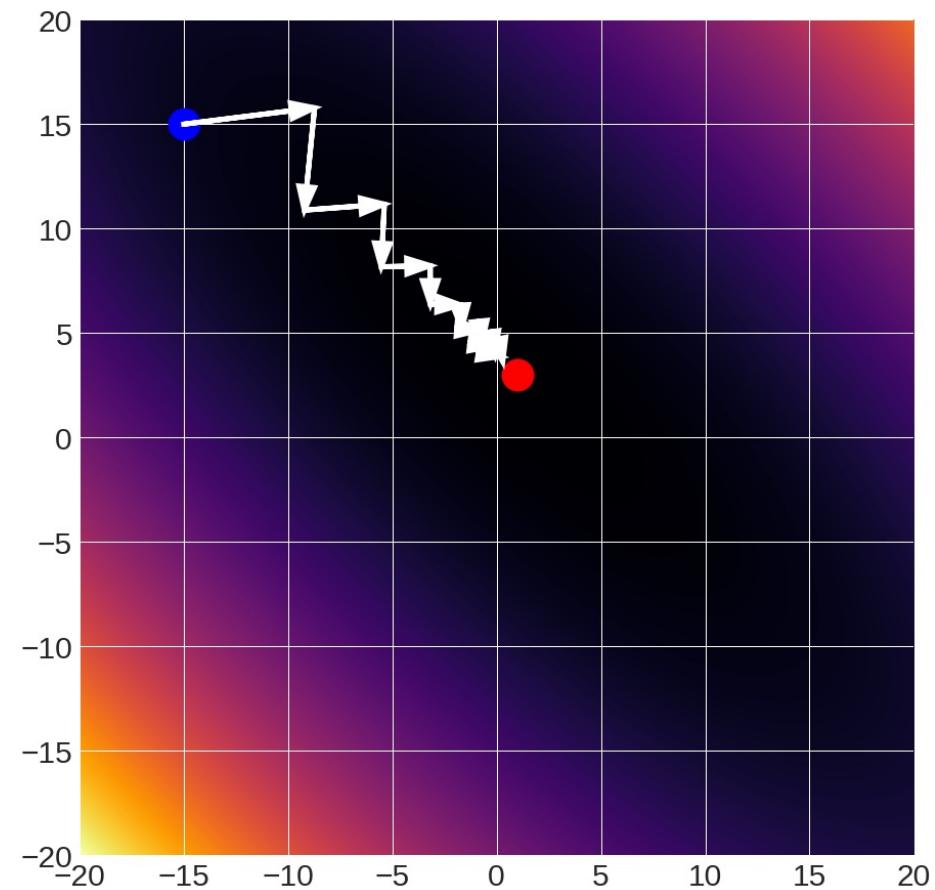
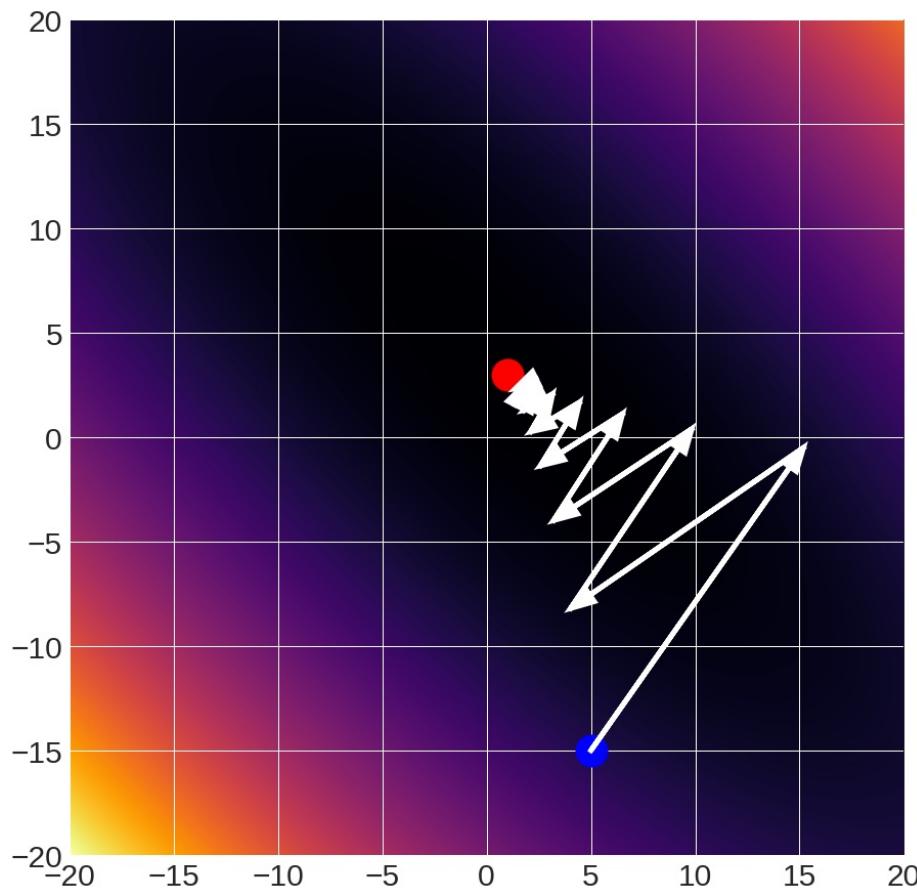
Method: at each step, move in direction of negative gradient

```
w0 = initialize() #initialize  
for iter in range(numIters):  
    g =  $\nabla_w L(w)$  #eval gradient  
    w = w + -stepsize(iter)*g #update w  
return w
```

Gradient Descent

Given starting point (blue)

$$w_{i+1} = w_i + -9.8 \times 10^{-2} \times \text{gradient}$$



Computing the Gradient

How Do You Compute The Gradient?
Numerical Method:

$$\nabla_w L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$

How do you compute this?

$$\frac{\partial f(x)}{\partial x} = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

In practice, use:

$$\frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}$$

Computing the Gradient

How Do You Compute The Gradient?
Numerical Method:

$$\nabla_w L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$

Use: $\frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}$

How many function evaluations per dimension?

For each dimension : 2 times.

All dimensions : $2 \times n = 2n$

Computing the Gradient

How Do You Compute The Gradient?
Analytical Method:

$$\nabla_w L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$

Use calculus!

Computing the Gradient

$$L(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

 $\frac{\partial}{\partial \mathbf{w}}$ 

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\lambda \mathbf{w} + \sum_{i=1}^n -(2(y_i - \mathbf{w}^T \mathbf{x}_i)\mathbf{x}_i)$$

Interpreting Gradients (1 Sample)

Recall:

$$\mathbf{w} = \mathbf{w} + -\nabla_{\mathbf{w}} L(\mathbf{w}) \text{ #update w}$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\lambda\mathbf{w} + -(2(y - \mathbf{w}^T \mathbf{x})\mathbf{x})$$

Push \mathbf{w} towards 0

$$-\nabla_{\mathbf{w}} L(\mathbf{w}) = -2\lambda\mathbf{w} + (2(y - \mathbf{w}^T \mathbf{x})\mathbf{x})$$

If $y > \underbrace{\mathbf{w}^T \mathbf{x}}$ (too low): then $\mathbf{w} = \mathbf{w} + \alpha \mathbf{x}$ for some α
little bit short

Before: $\mathbf{w}^T \mathbf{x}$

After: $(\mathbf{w} + \alpha \mathbf{x})^T \mathbf{x} = \mathbf{w}^T \mathbf{x} + \underbrace{\alpha \mathbf{x}^T \mathbf{x}}_{\text{positive}}$
make it bigger.

Quick annoying detail: subgradients

What is the derivative of $|x|$?

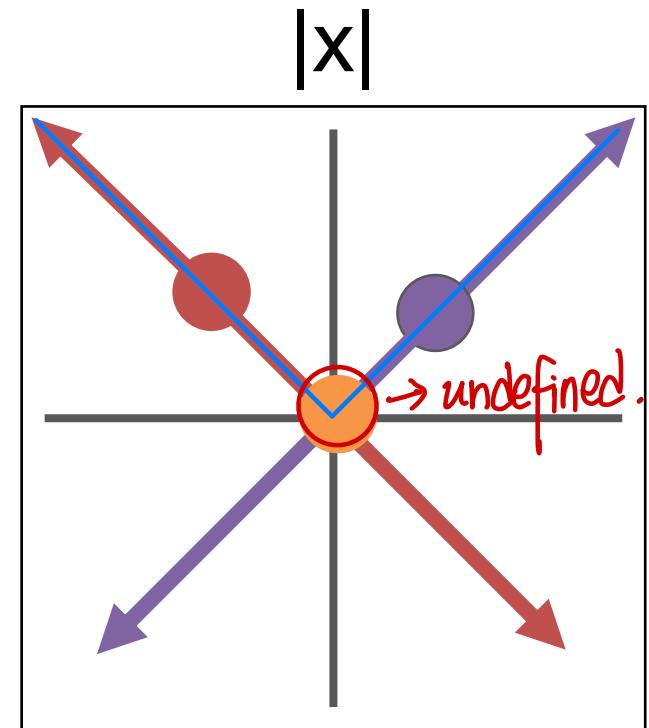
Derivatives/Gradients

Defined everywhere but 0

$$\frac{\partial}{\partial x} f(x) = \text{sign}(x) \quad x \neq 0$$

undefined

$x = 0$



Oh no! A discontinuity!

Quick annoying detail: subgradients

Subgradient: any underestimate of function

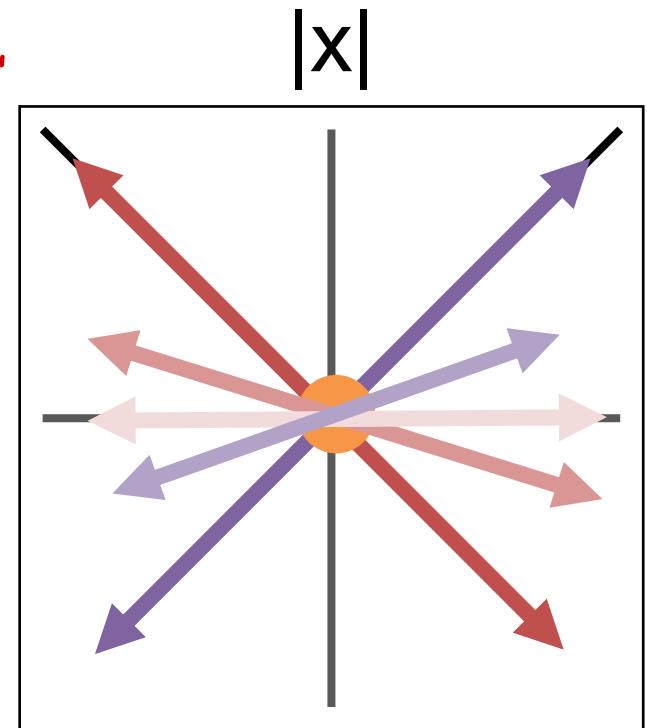
Subderivatives/subgradients

Defined everywhere.

Defined everywhere

$$\frac{\partial}{\partial x} f(x) = \text{sign}(x) \quad x \neq 0$$

$$\frac{\partial}{\partial x} f(x) \in [-1,1] \quad x = 0$$



In practice: at discontinuity, pick value on either side.

Computing The Gradient

- Numerical: foolproof but slow
- Analytical: can mess things up ☺
- In practice: do analytical, but check with numerical (called a gradient check)

Implementing Gradient Descent

Loss is a function that we can evaluate over data

All
Data

$$-\nabla_{\mathbf{w}} L(\mathbf{w}) = -2\lambda \mathbf{w} + \sum_{i=1}^n (2(y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i)$$

Subset
B

$$-\nabla_{\mathbf{w}} L_B(\mathbf{w}) = -2\lambda \mathbf{w} + \sum_{i \in B} (2(y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i)$$

Implementing Gradient Descent

Option 1: Vanilla Gradient Descent

Compute gradient of L over all data points

```
for iter in range(numIters):
```

```
    g = gradient(data,L)
```

```
    w = w + -stepsize(iter)*g #update w
```

Implementing Gradient Descent

Option 2: *Stochastic* Gradient Descent

Compute gradient of L over 1 random sample

```
for iter in range(numIters):  
    index = randint(0,#data)  
    g = gradient(data[index],L)  
    w = w + -stepSize(iter)*g #update w
```

Implementing Gradient Descent

Option 3: *Minibatch* Gradient Descent

Compute gradient of L over subset of B samples

```
for iter in range(numIters):  
    subset = choose_samples(#data,B)  
    g = gradient(data[subset],L)  
    w = w + -stepsize(iter)*g #update w
```

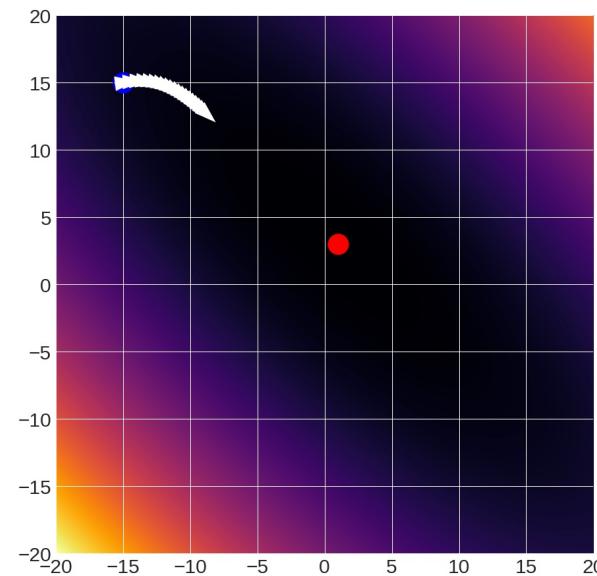
Typical batch sizes: ~100 (although there's lots of great research on huge batch sizes, e.g., *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*)

Gradient Descent Details

Step size (also called **learning rate / lr**)
critical parameter

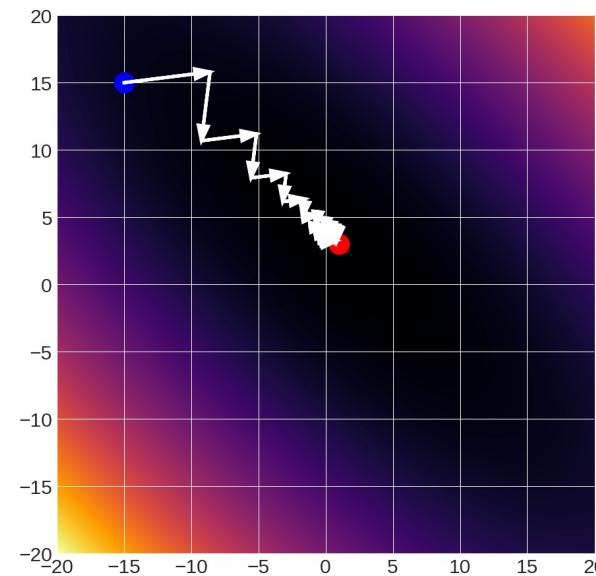
1×10^{-2}

falls short



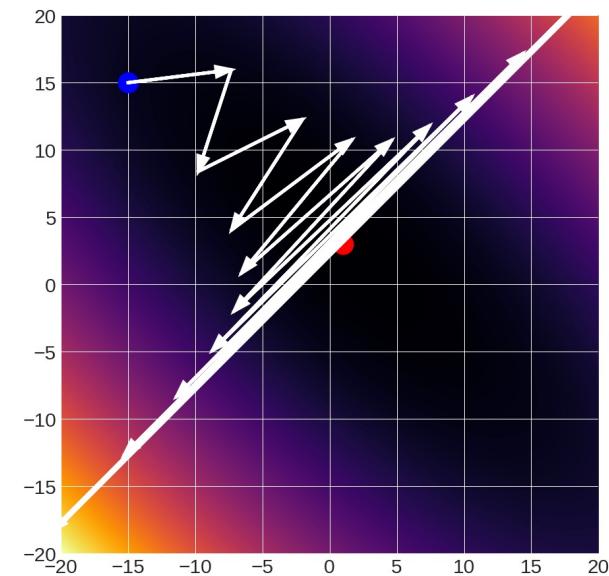
10×10^{-2}

converges



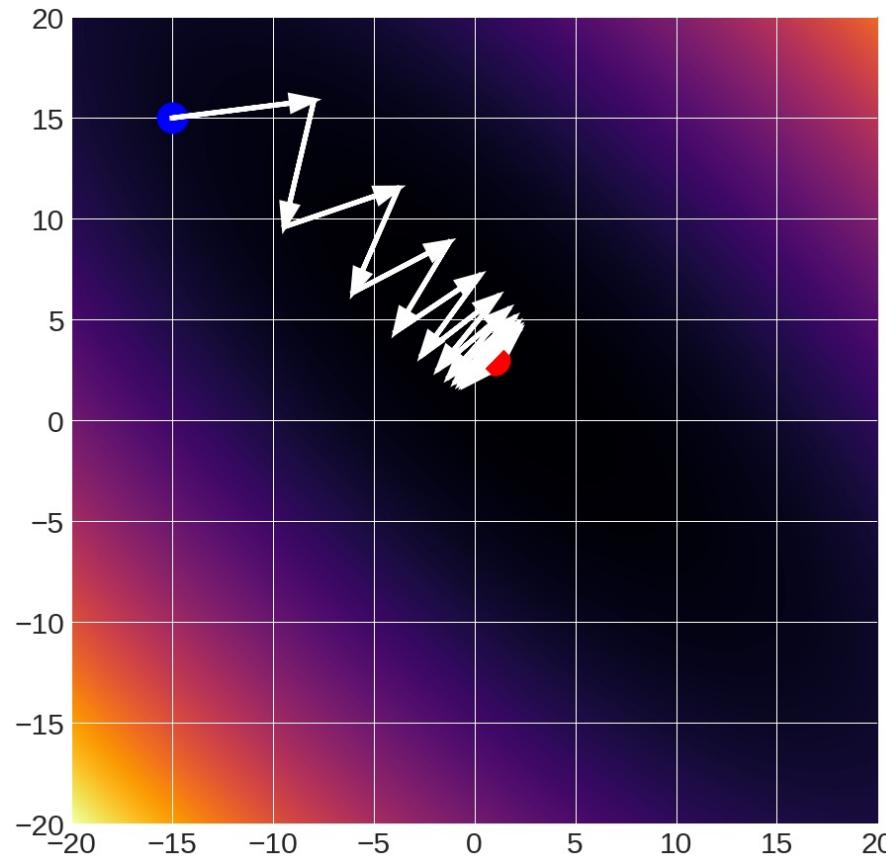
12×10^{-2}

diverges



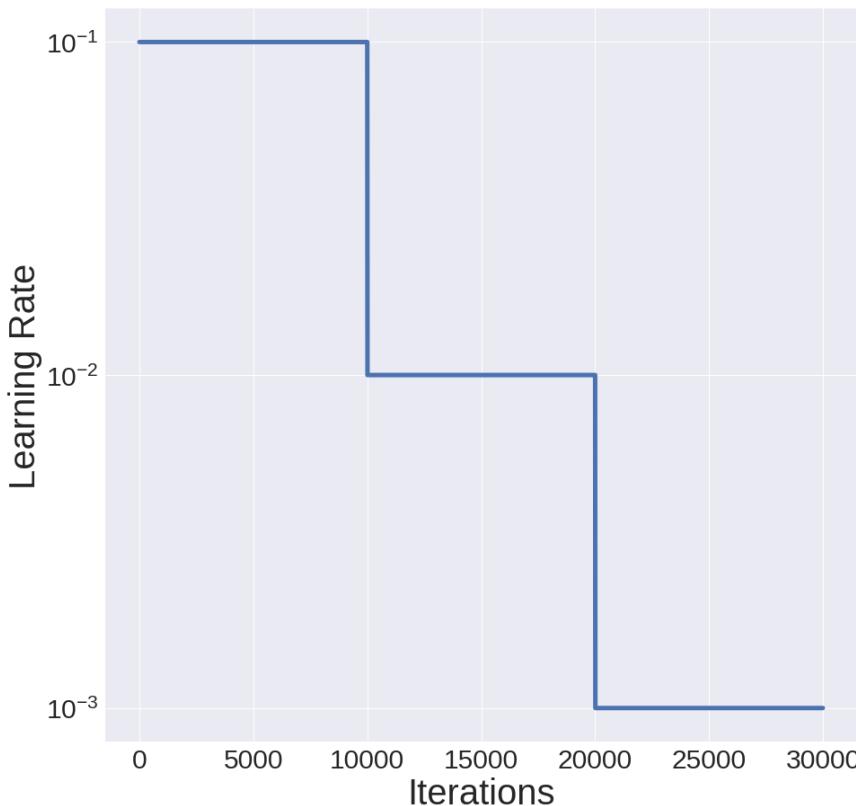
Gradient Descent Details

11×10^{-2} : oscillates
(Visualizing the Raw gradients)



Gradient Descent Details

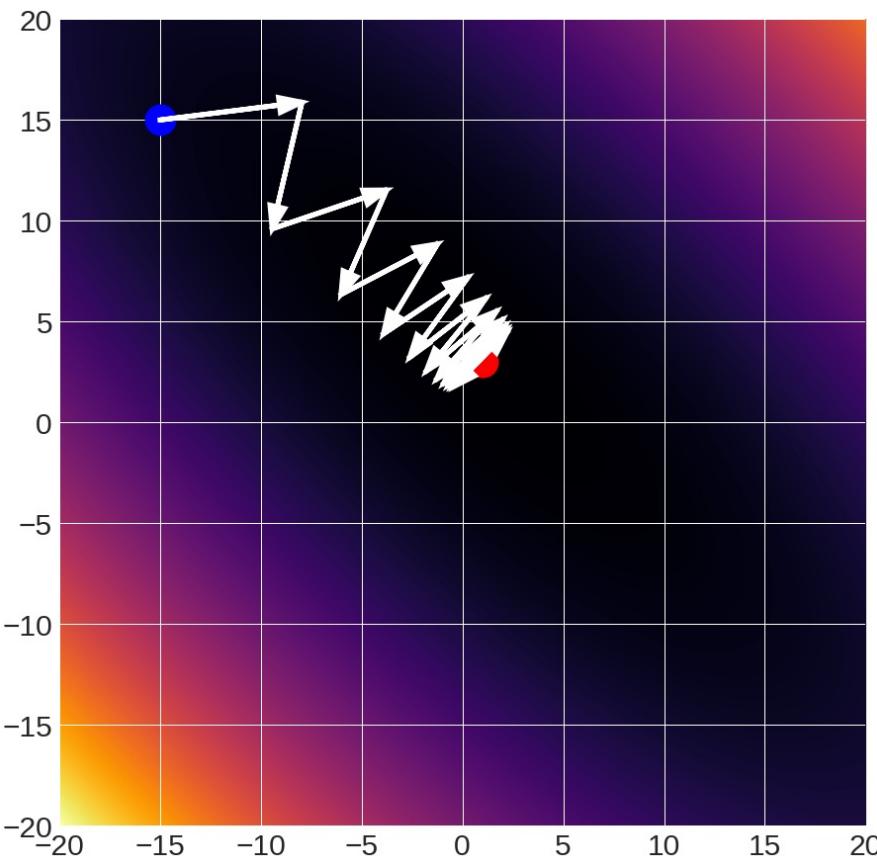
One solution: start with initial rate lr ,
multiply by f every N iterations



$init_lr = 10^{-1}$
 $f = 0.1$
 $N = 10K$

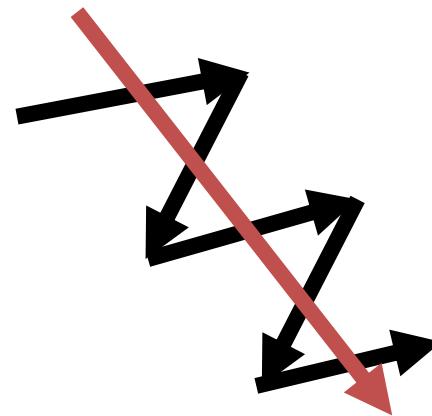
Gradient Descent Details

11×10^{-2} : oscillates
(Raw gradients)



Another Solution:
Average gradients

With exponentially decaying
weights, called “momentum”

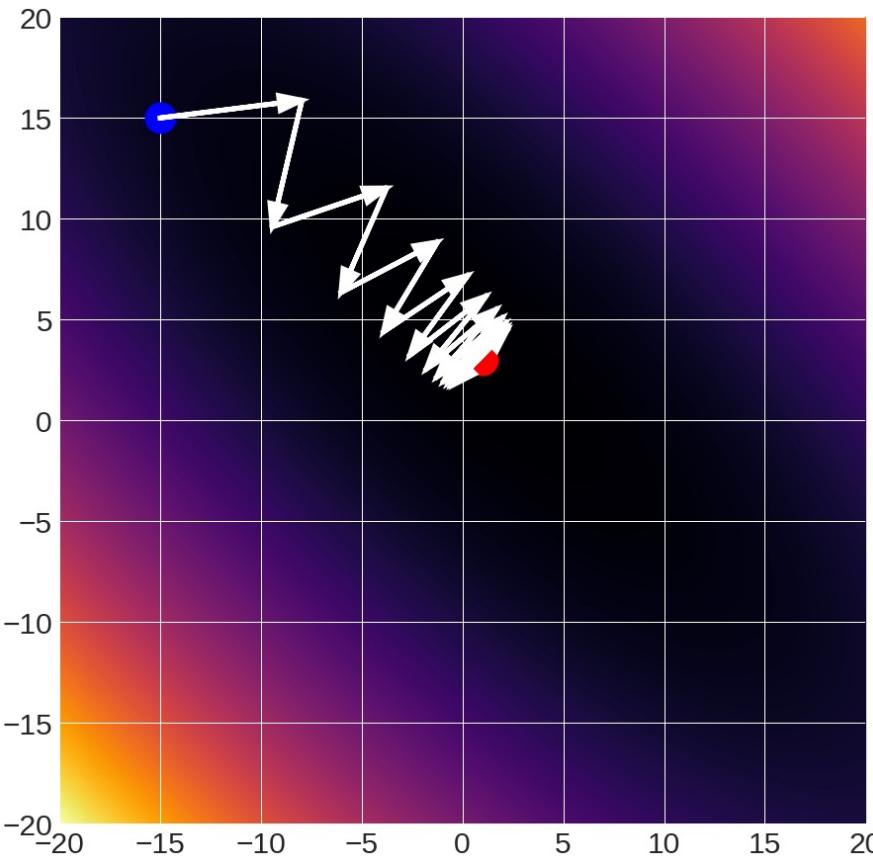


$$m_t = \beta \cdot m_{t-1} + (1 - \beta) \nabla_w L(\mathbf{w})$$

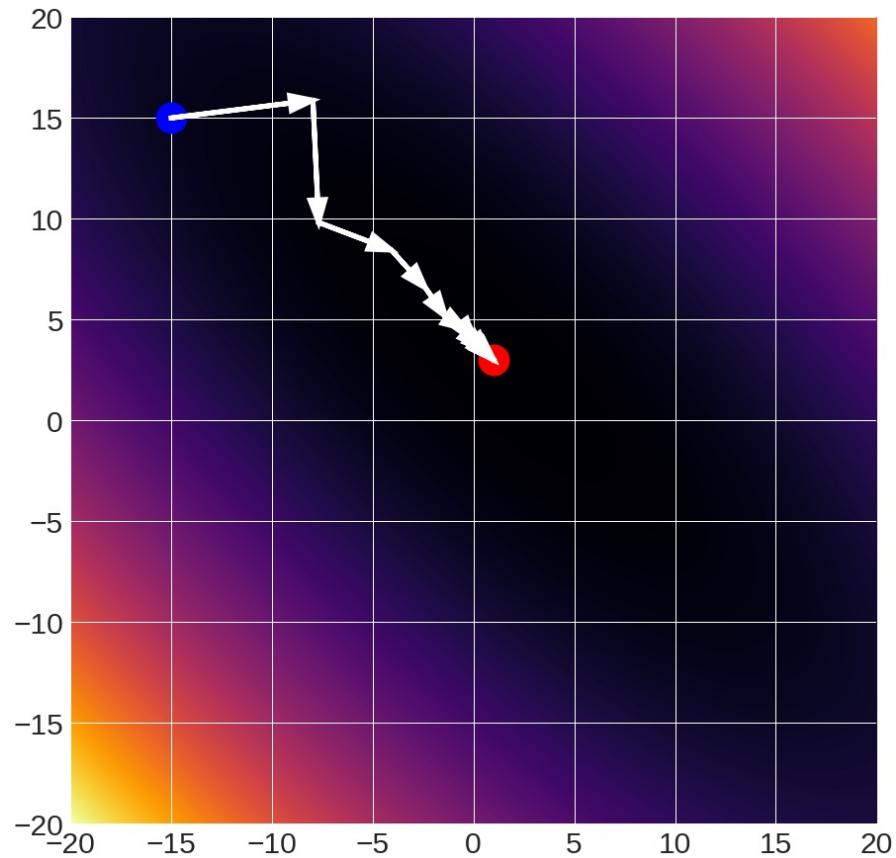
Use m_t instead of $\nabla_w L(\mathbf{w})$

Gradient Descent Details

11×10^{-2} : oscillates
(Raw gradients)



11×10^{-2}
(0.25 momentum)

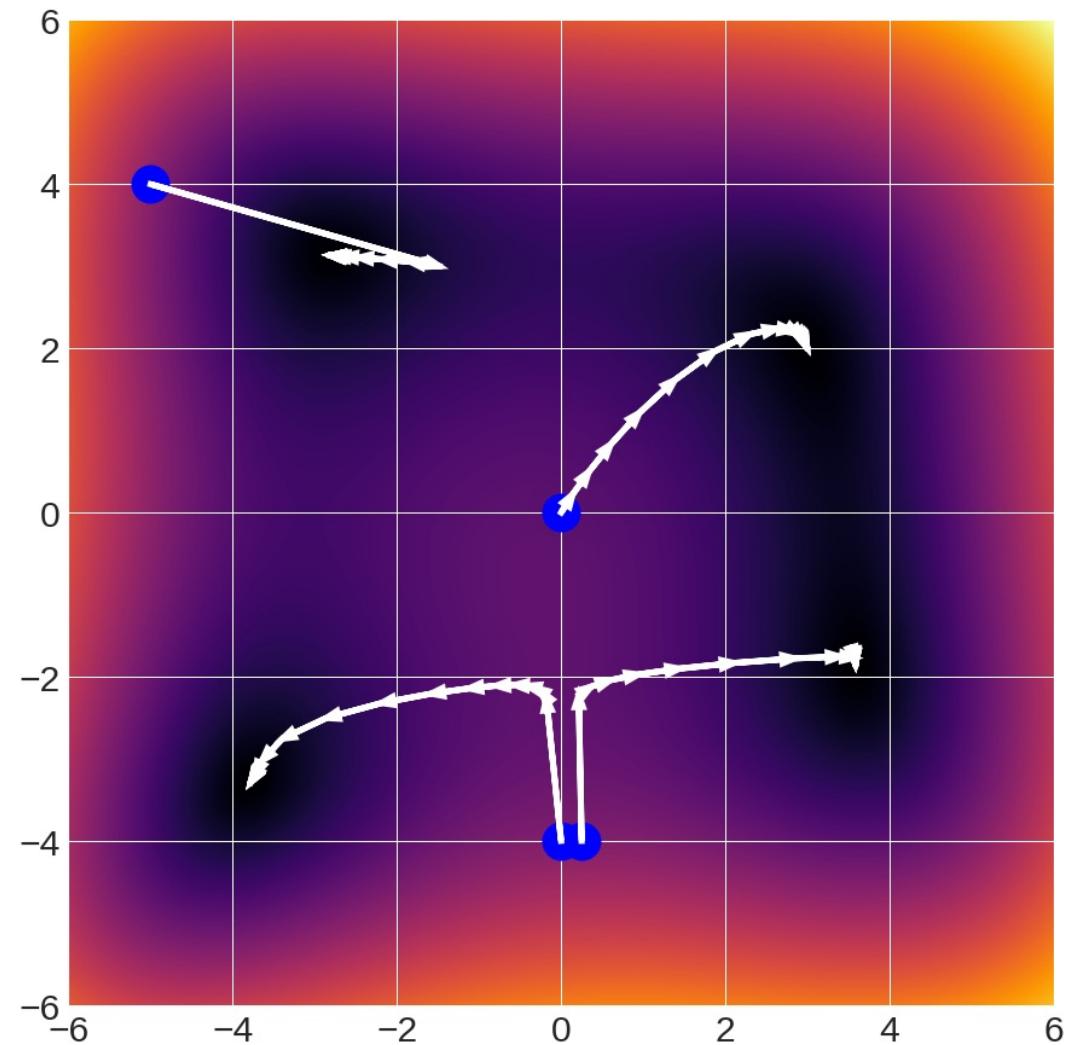


Gradient Descent Details

Multiple Minima

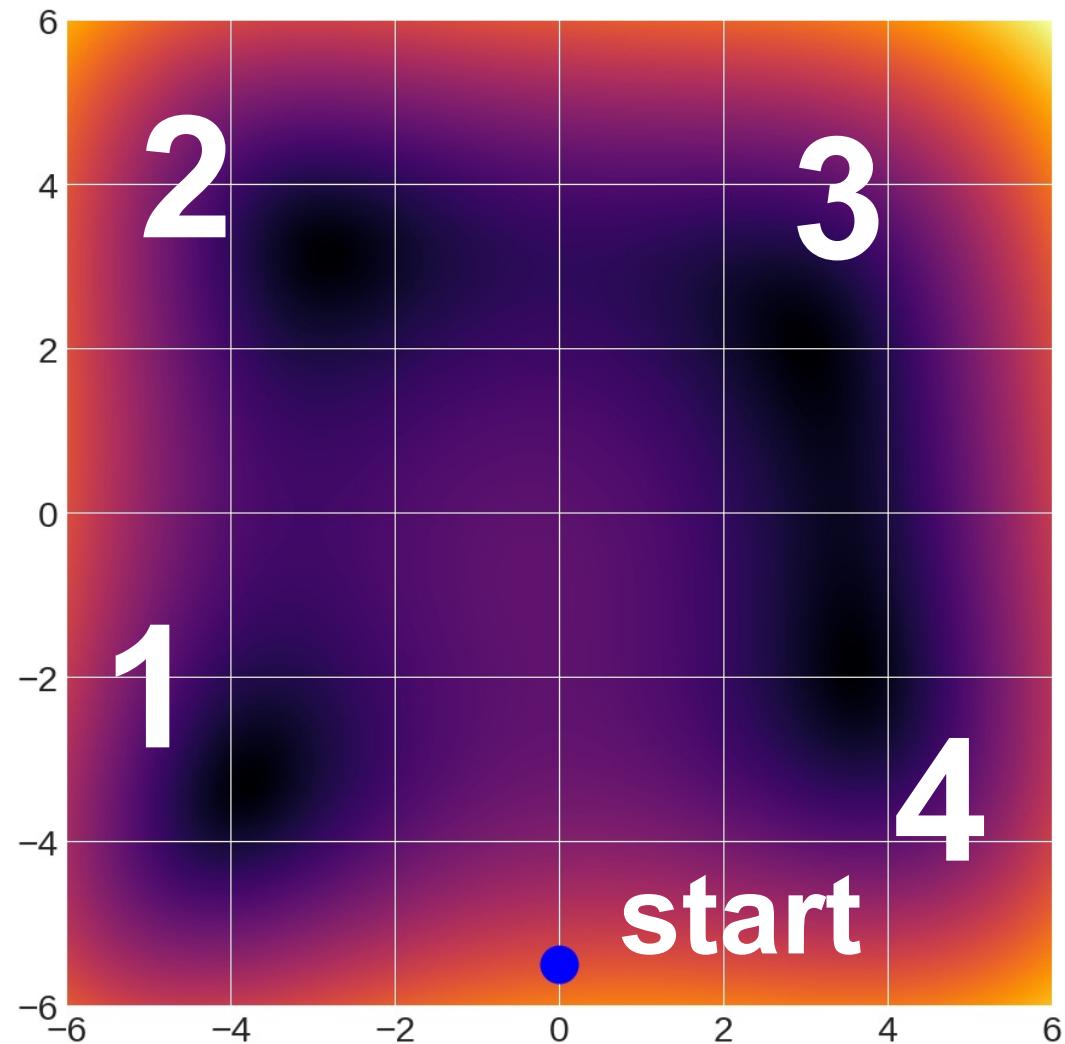


Gradient Descent
Finds **local**
minimum

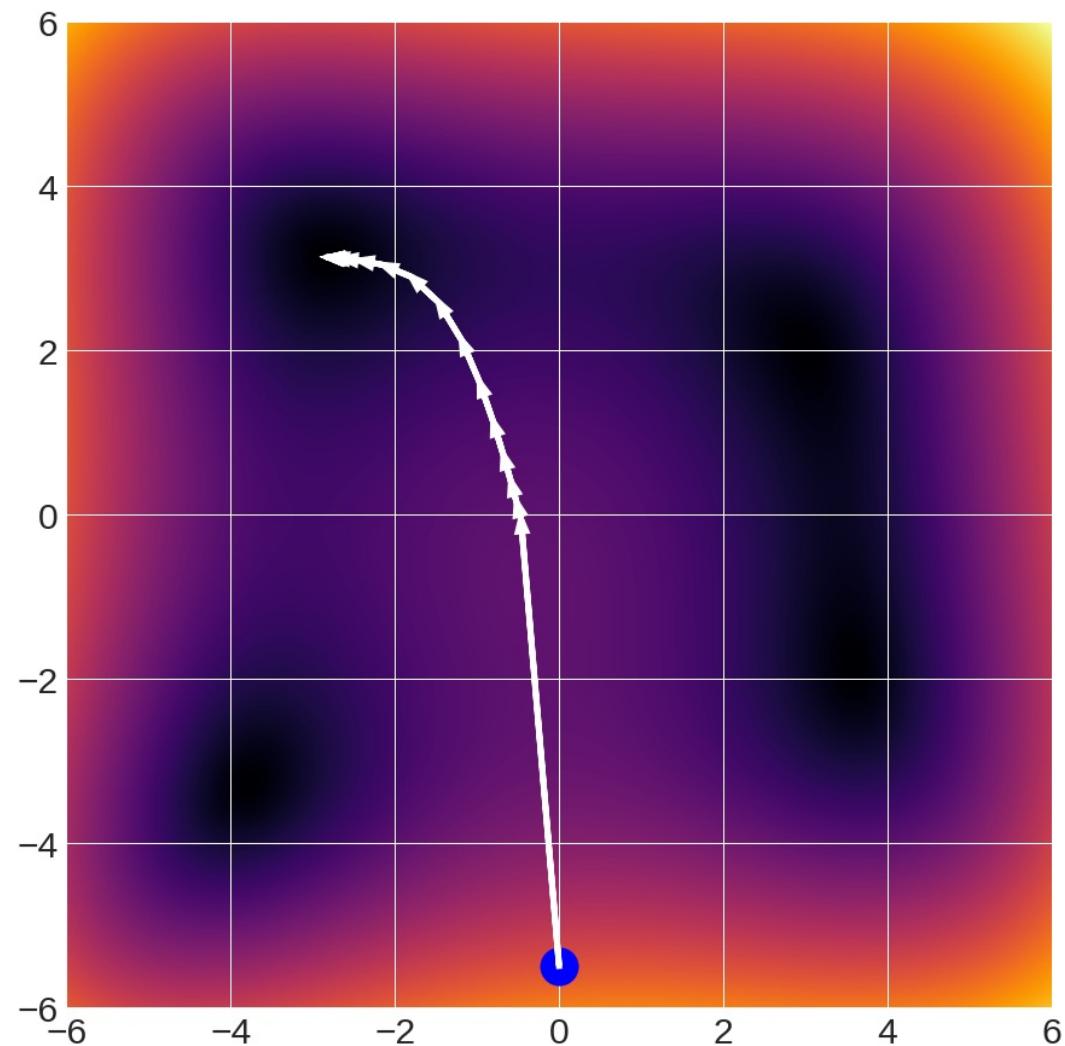


Gradient Descent Details

Guess the
minimum!
(where will it
converge?)

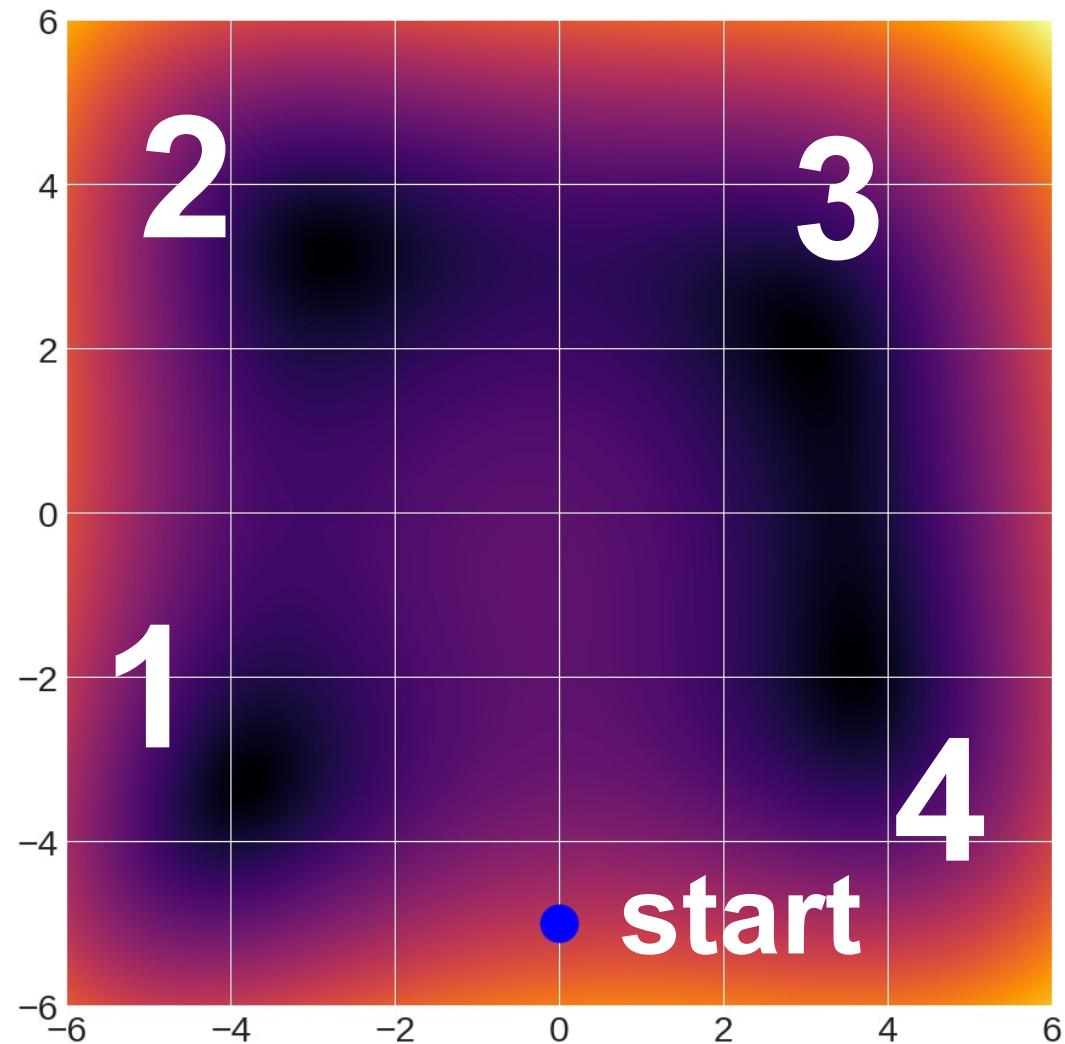


Gradient Descent Details



Gradient Descent Details

Guess the
minimum!
(where will it
converge?)

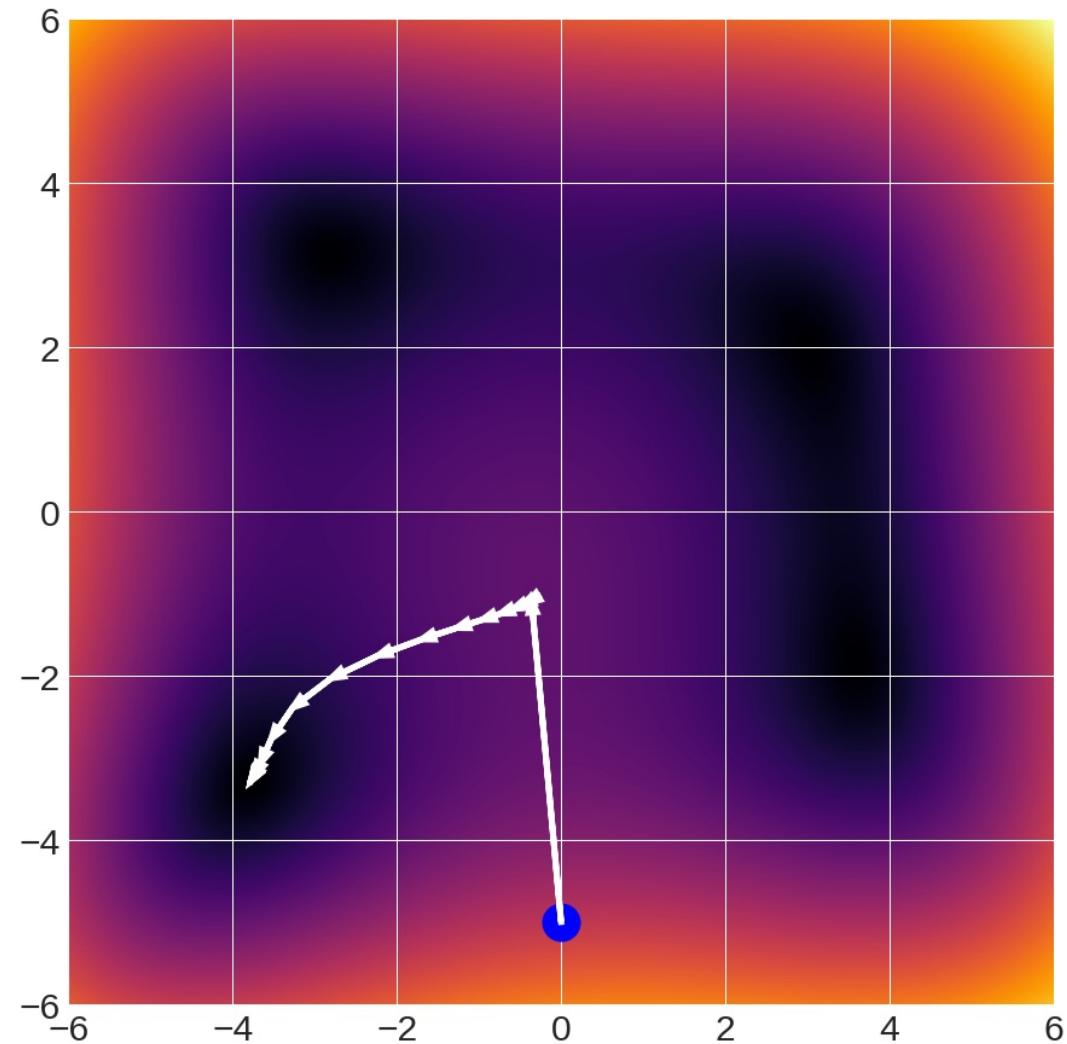


Gradient Descent Details

Dynamics are fairly complex

Many important functions are **convex**: any local minimum is a global minimum

Many important functions are not.



In practice

- **Conventional wisdom:** minibatch stochastic gradient descent (SGD) + momentum (package implements it for you, e.g., Adam) + some sensibly changing learning rate
- The above is typically what is meant by “SGD”
- Other update rules exist (e.g., AdamW, LAMB). Can often work better on some problems.

Optimizing Everything

$$L(\mathbf{W}) = \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n -\log \left(\frac{\exp((\mathbf{W}\mathbf{x})_{y_i})}{\sum_k \exp((\mathbf{W}\mathbf{x})_k)} \right)$$

$$L(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- Optimize \mathbf{w} on training set with SGD to maximize training accuracy
- Optimize λ with random/grid search to maximize validation accuracy.
- **Why not gradient descent λ ?**
- Note: Optimizing λ on training sets it to 0

(Over/Under)fitting and Complexity

Let's fit a polynomial: given x, predict y

Note: can do non-linear regression with copies of x

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_1^F & \cdots & x_1^2 & x_1 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_N^F & \cdots & x_N^2 & x_N & 1 \end{bmatrix} \begin{bmatrix} w_F \\ \vdots \\ w_2 \\ w_1 \\ w_0 \end{bmatrix}$$

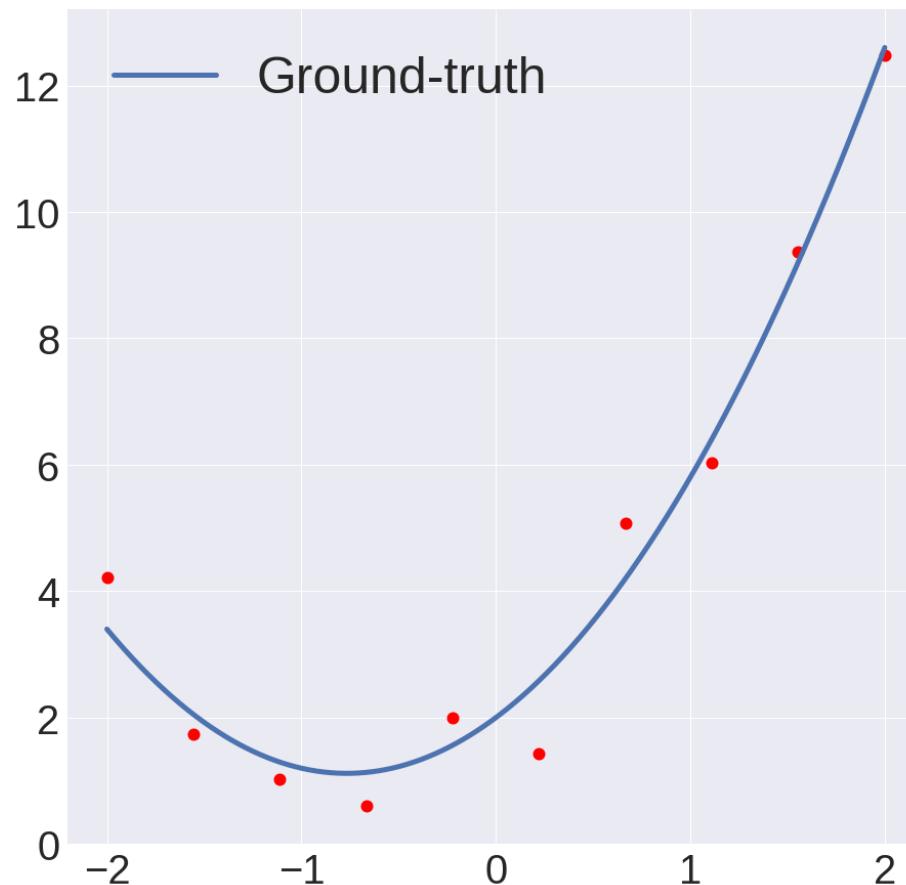
Matrix of all polynomial degrees ↑

Weights: one per polynomial degree ↑

Abel-Ruffini Theorem: There is no general solution to polynomial equations of degree five or higher

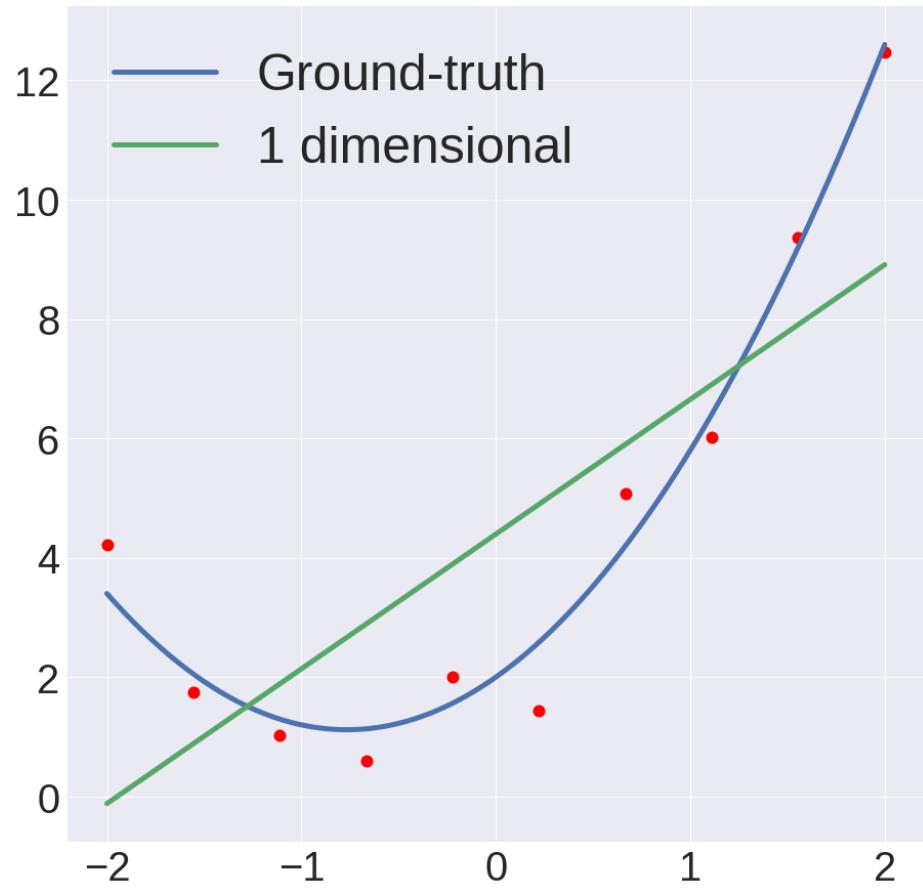
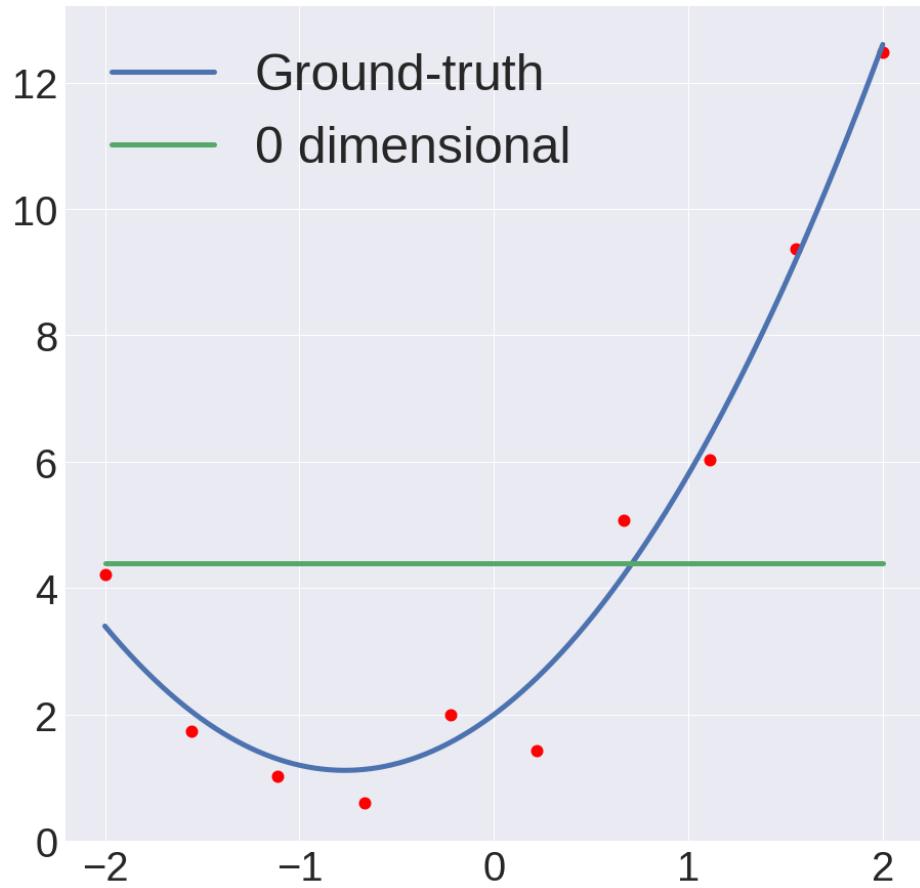
(Over/Under)fitting and Complexity

Model: $1.5x^2 + 2.3x + 2 + N(0, 0.5)$

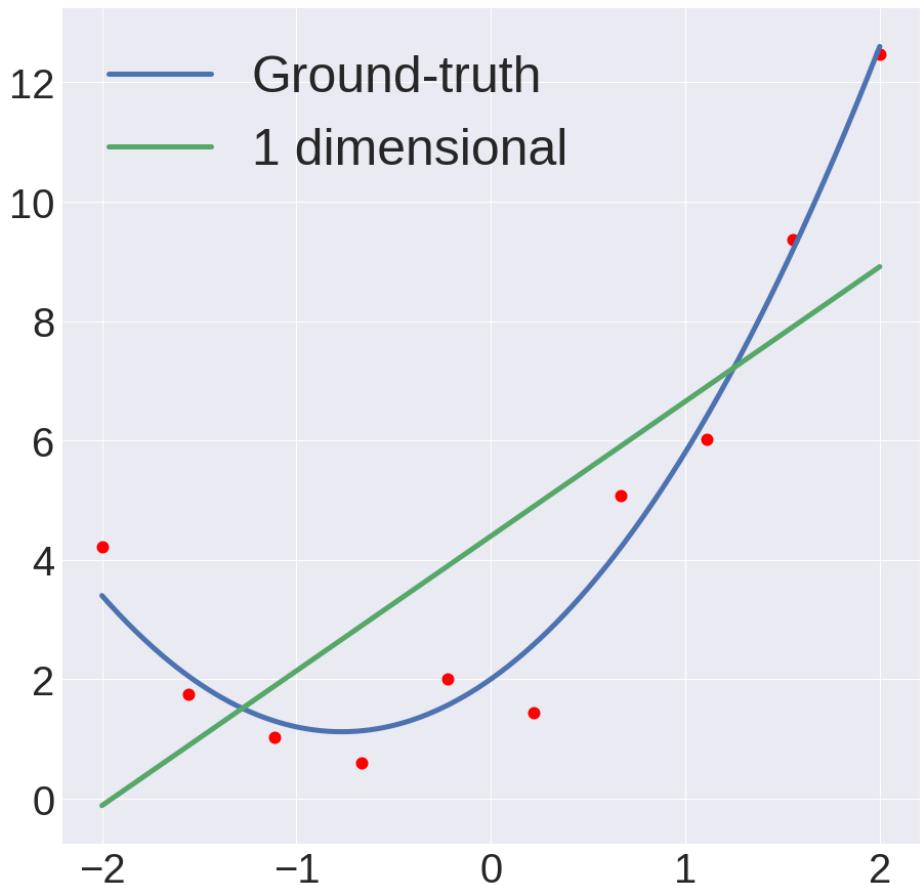


Underfitting

Model: $1.5x^2 + 2.3x + 2 + N(0, 0.5)$



Underfitting

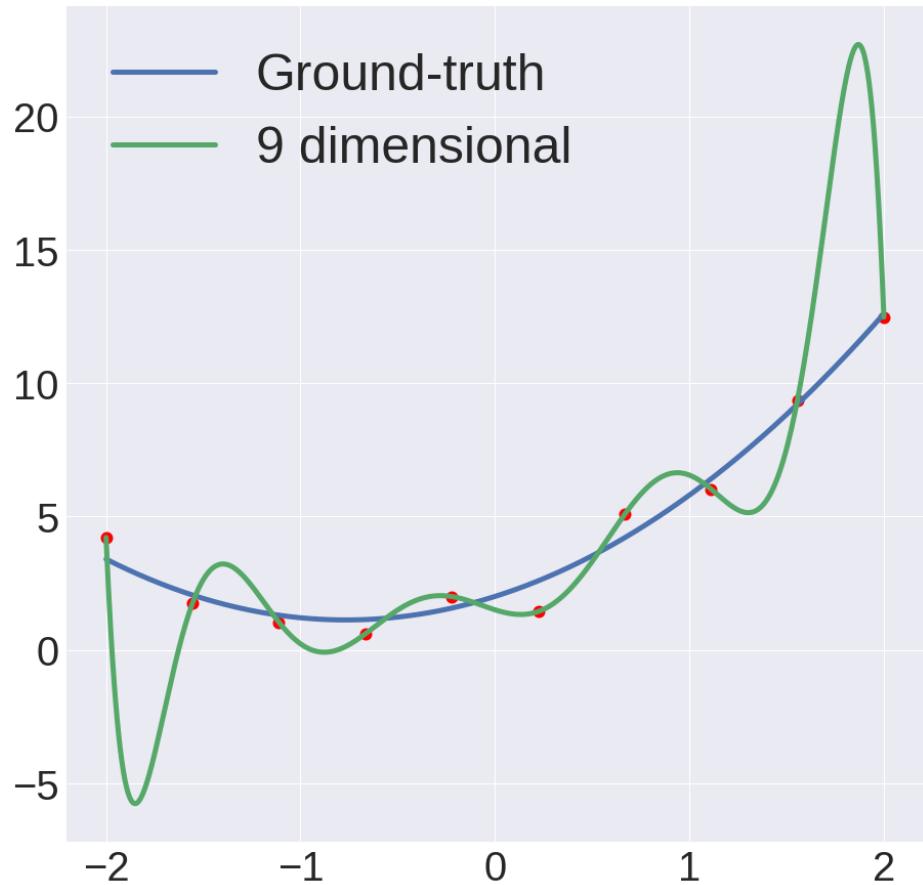
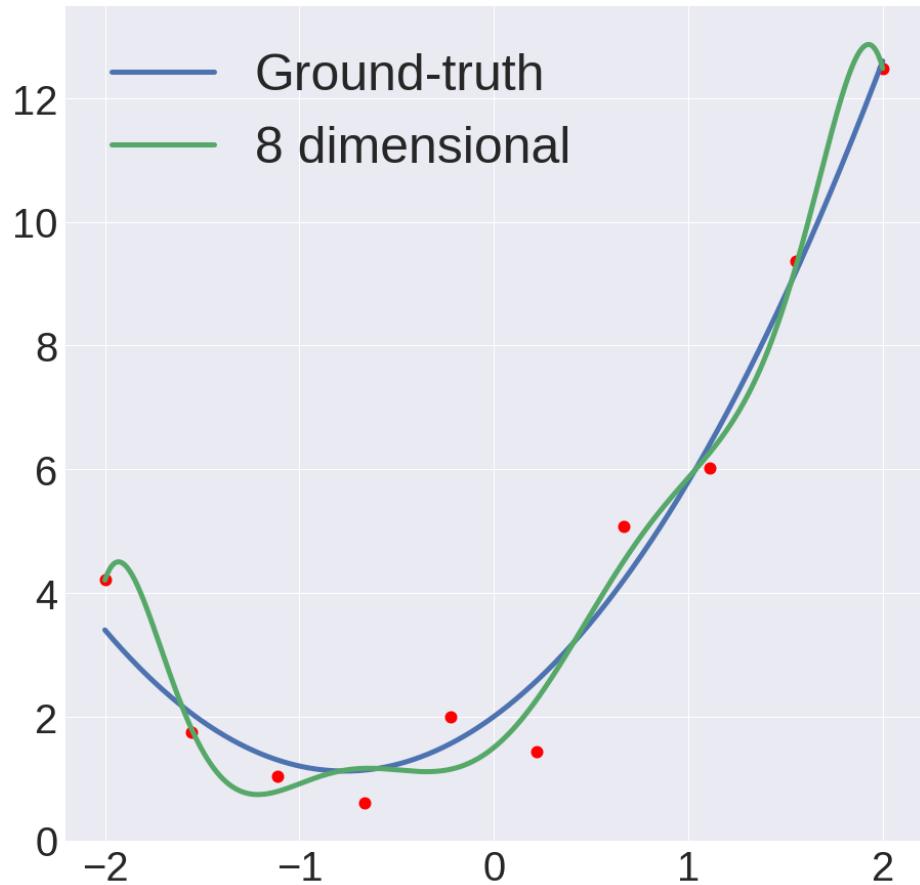


Model doesn't have the parameters to fit the data.

Bias (statistics): Error intrinsic to the model.

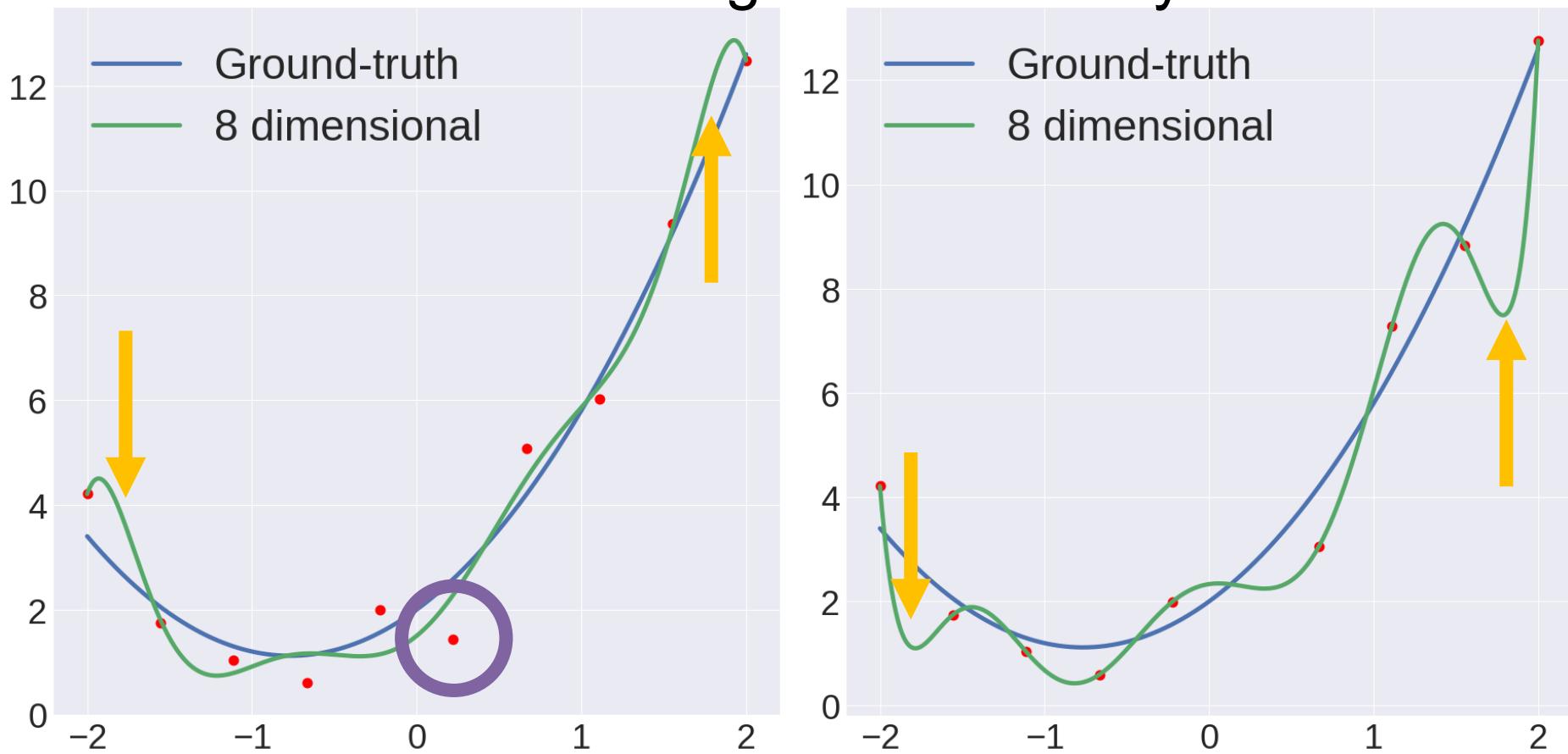
Overfitting

Model: $1.5x^2 + 2.3x + 2 + N(0, 0.5)$



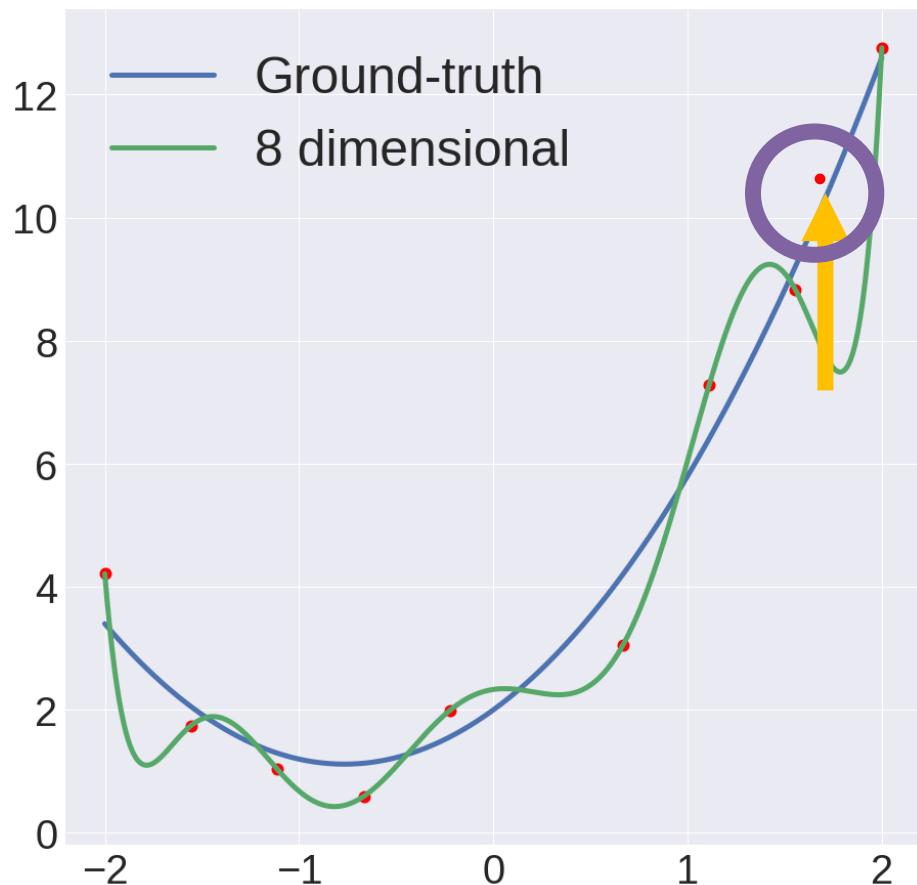
Overfitting

Model has high ***variance***: remove **one point**, and model changes dramatically



Overfitting

Model has high ***variance***: add **one unseen point**, and model prediction accuracy drastically changes



(Continuous) Model Complexity

$$\arg \min_W \lambda \|W\|_2^2 + \sum_{i=1}^n -\log \left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)} \right)$$

Regularization: penalty
for complex model

Pay penalty for negative log-
likelihood of correct class

Intuitively: big weights = more complex model

Model 1: $0.01*x_1 + 1.3*x_2 + -0.02*x_3 + -2.1*x_4 + 10$

Model 2: $37.2*x_1 + 13.4*x_2 + 5.6*x_3 + -6.1*x_4 + 30$

since the weight is bigger \longrightarrow the \bar{x} different can cause bigger difference.
Why? (too sensitive to the input).

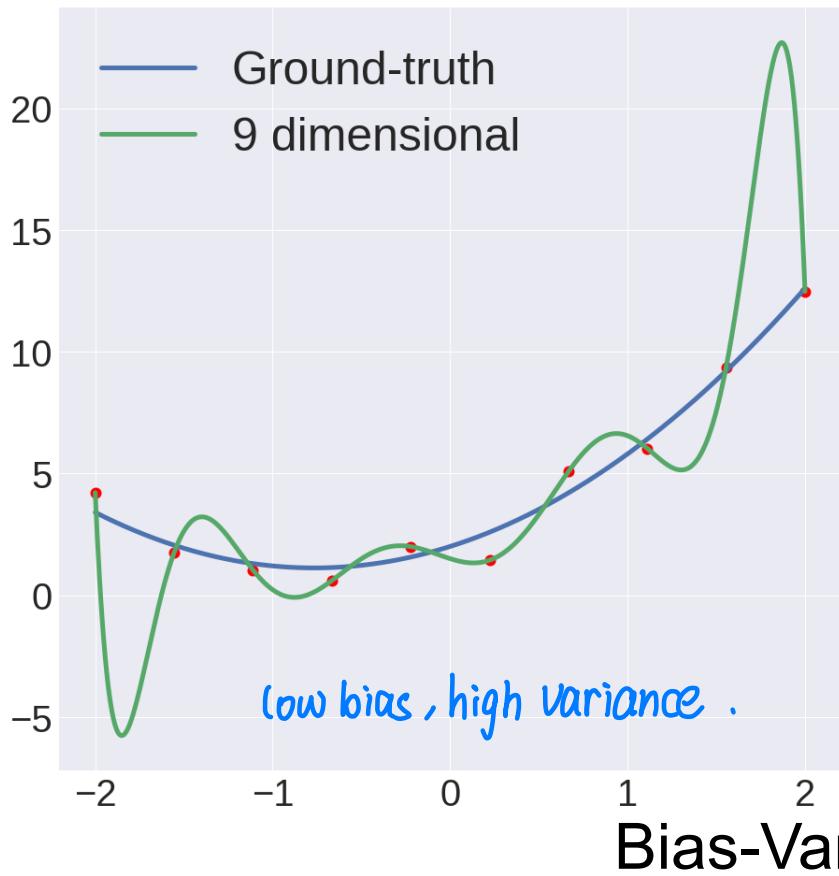
Fitting Model

Again, fitting polynomial, but with regularization

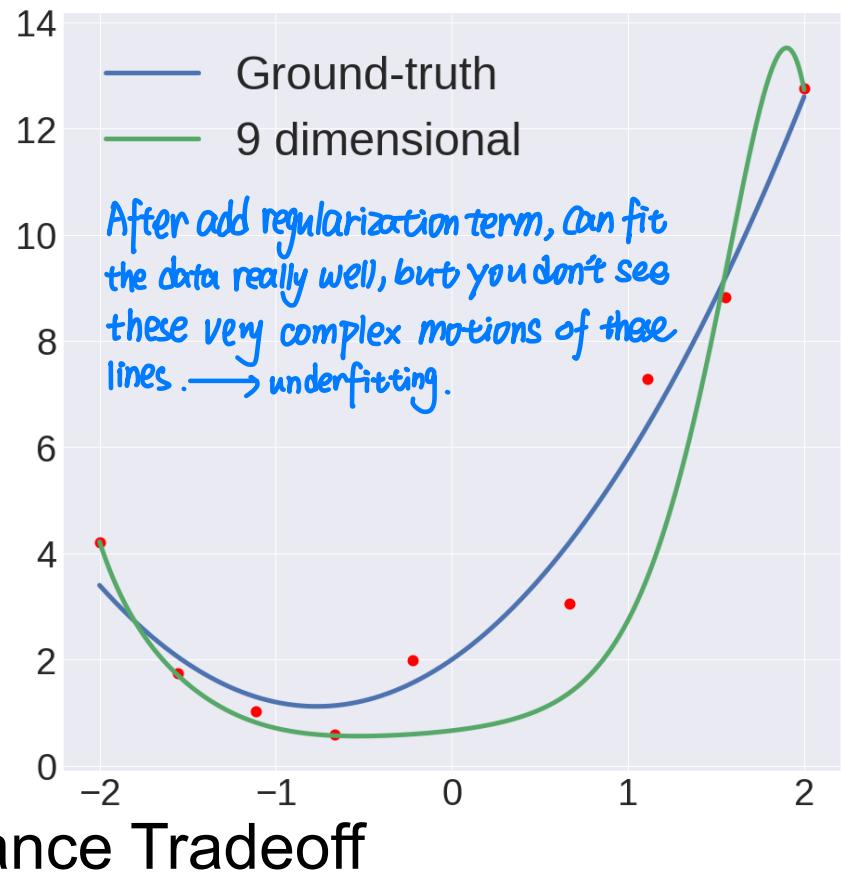
$$\arg \min_w \|y - Xw\|_2^2 + \lambda \|w\|_2^2$$
$$\begin{bmatrix} x_1^F & \cdots & x_1^2 & x_1 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_N^F & \cdots & x_N^2 & x_N & 1 \end{bmatrix} \quad \begin{bmatrix} w_F \\ \vdots \\ w_0 \end{bmatrix}$$

Adding Regularization

No regularization:
fits all data points



Regularization:
can't fit all data points



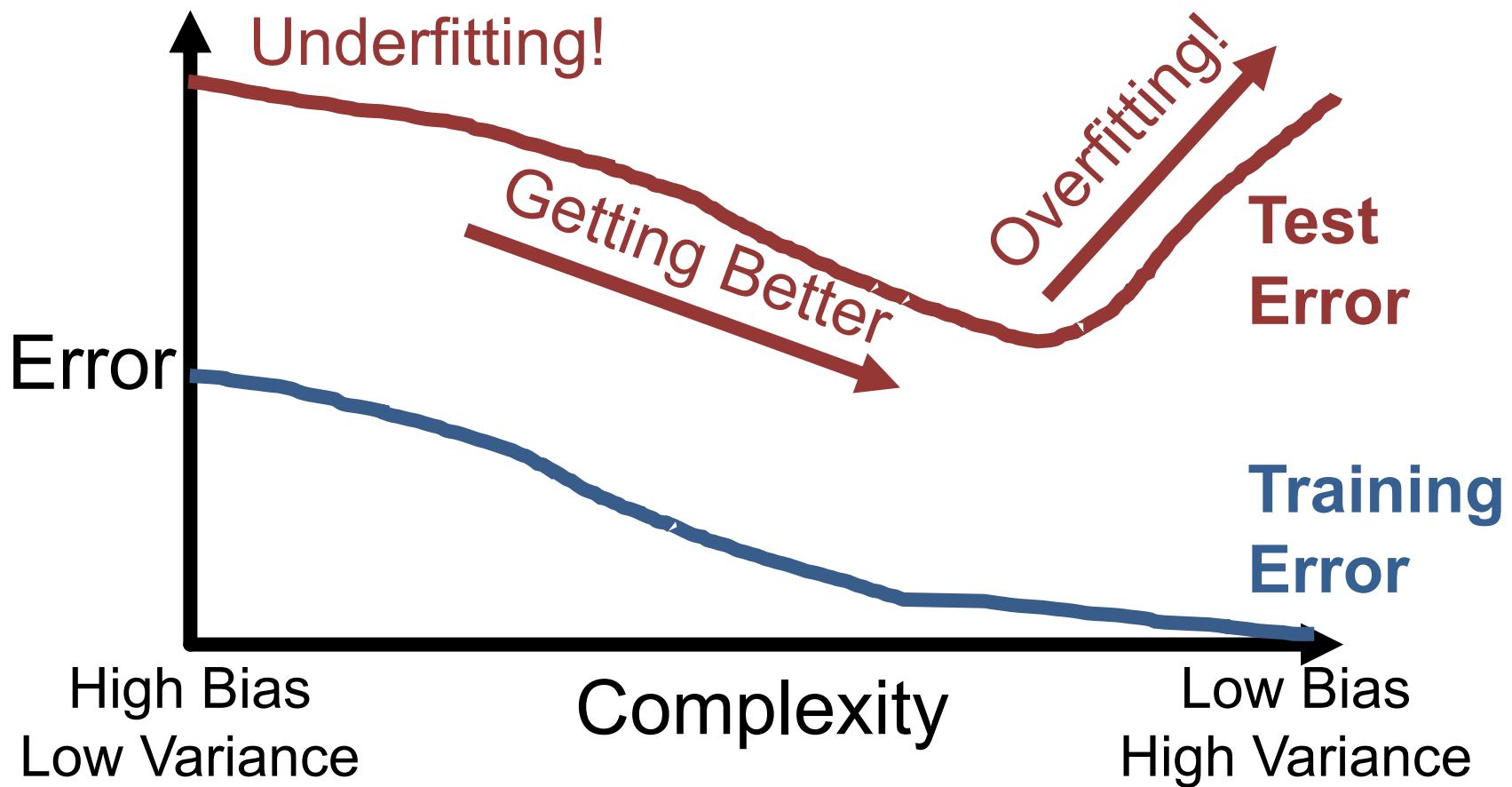
In General

Error on new data comes from combination of:

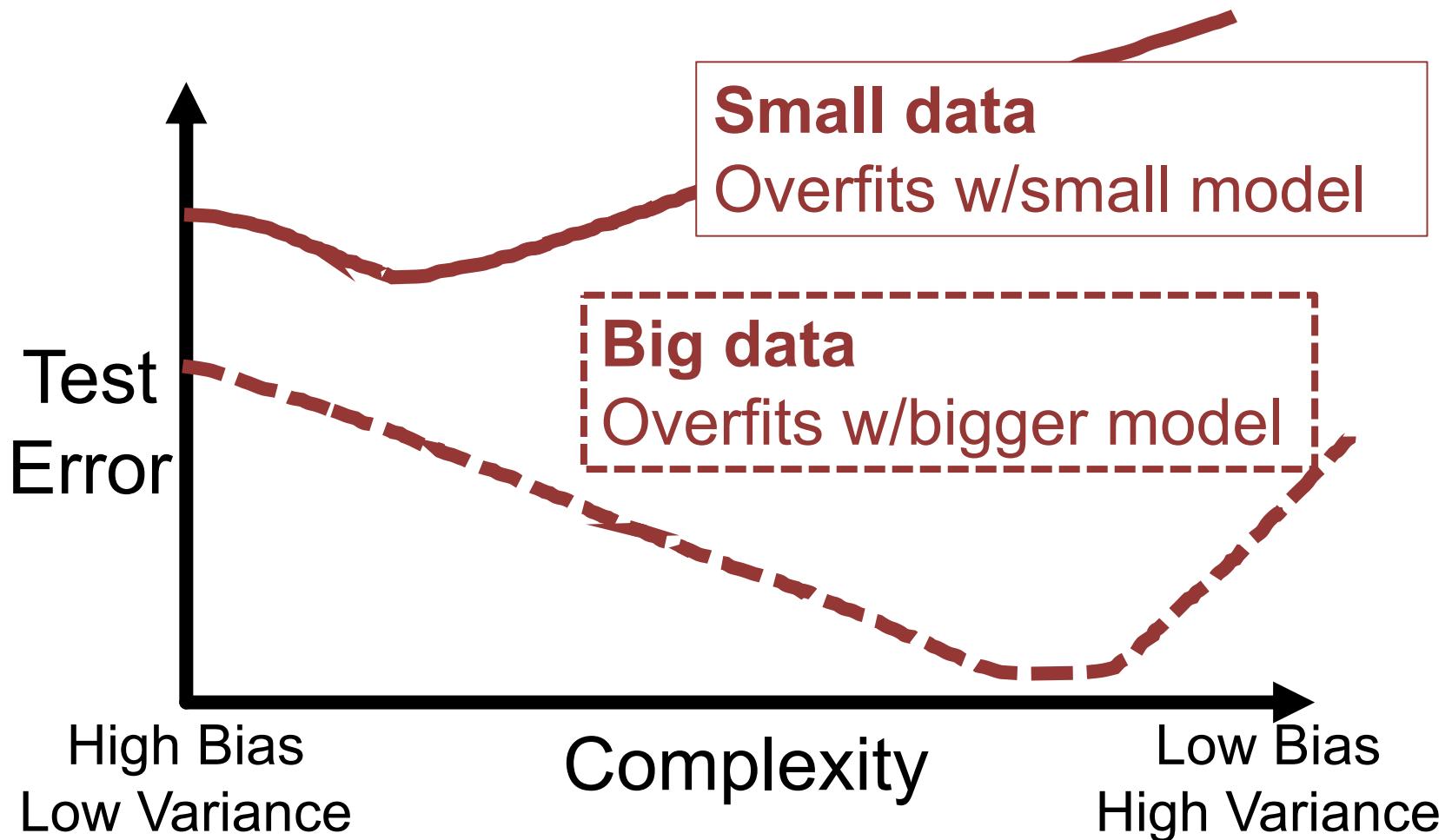
1. Bias: model is oversimplified and can't fit the underlying data
2. Variance: you don't have the ability to estimate your model from limited data
3. Inherent: the data is intrinsically difficult (Pattern too complex)

Bias and variance trade-off. Fixing one hurts the other.

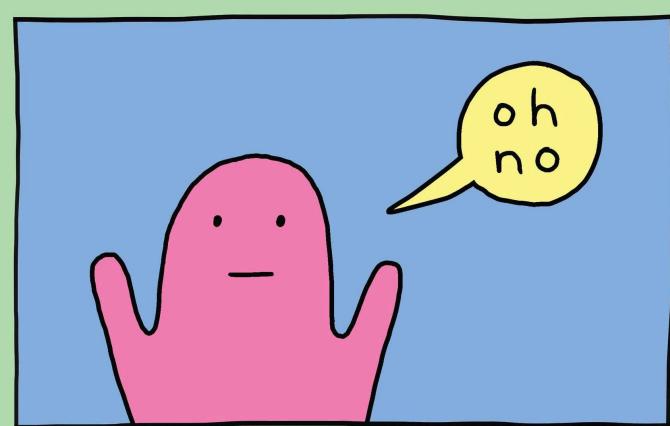
Underfitting and Overfitting



Underfitting and Overfitting



Underfitting and Overfitting



alex norris

Small data

Lots of this behavior
doesn't seem to
replicate in models with
high learning capacity.

See below for details.

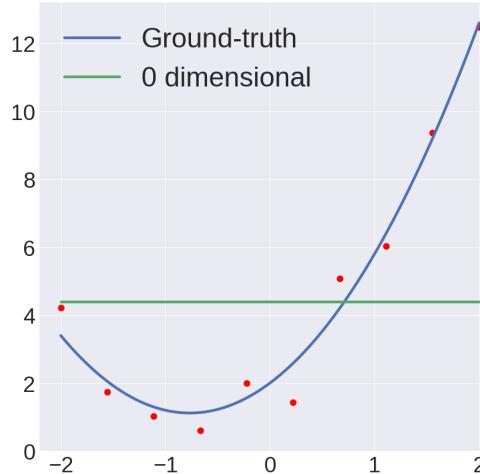
Low variance

High variance

Diagram adapted from: D. Hoiem; comic: <https://webcomicname.com/>

Reconciling modern machine learning practice and the bias-variance trade-off, Belkin et al. <https://arxiv.org/pdf/1812.11118.pdf>

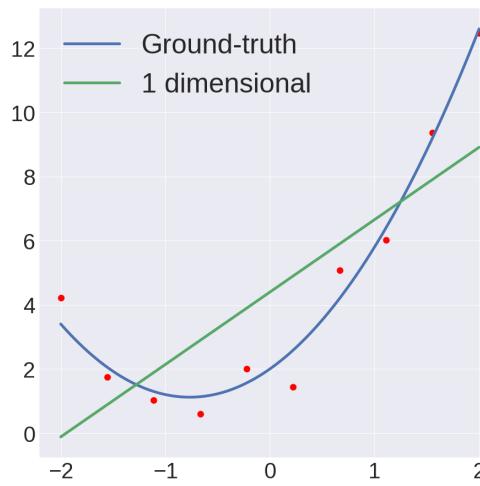
Underfitting



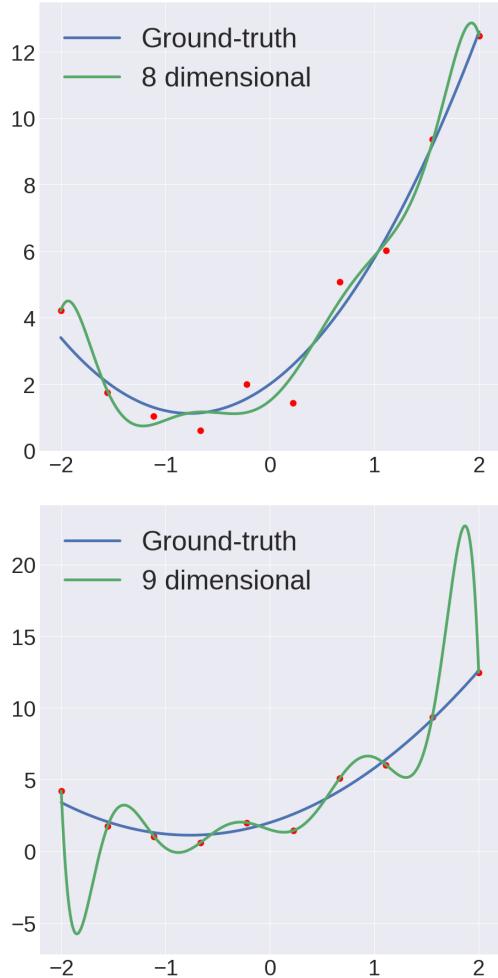
Do poorly on both training and validation data due to bias.

Solution:

1. More features
2. More powerful model
3. Reduce regularization



Overfitting



Do well on training data, but poorly on validation data due to variance

Solution:

1. More data
2. Less powerful model
3. Regularize your model more

Cris Dima rule: first make sure you can overfit, then stop overfitting.

Next Class

- Non-linear models (neural nets)

Let's Compute Another Gradient

- Below is another derivation that's worth looking at on your own time if you're curious

Computing The Gradient

Multiclass Support Vector Machine

$$\arg \min_{\mathbf{W}} \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n \sum_{j \neq y_i} \max(0, (\mathbf{W}\mathbf{x}_i)_j - (\mathbf{W}\mathbf{x}_i)_{y_i} + m)$$

Notation:

$\mathbf{W} \rightarrow$ rows \mathbf{w}_i (i.e., per-class scorer)

$(\mathbf{W}\mathbf{x}_i)_j \rightarrow \mathbf{w}_j^T \mathbf{x}_i$

$$\arg \min_{\mathbf{W}} \lambda \sum_{j=1}^K \|\mathbf{w}_j\|_2^2 + \sum_{i=1}^n \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T \mathbf{x}_i - \mathbf{w}_{y_i}^T \mathbf{x}_i + m)$$

Computing The Gradient

$$\arg \min_{\mathbf{W}} \lambda \sum_{j=1}^K \|\mathbf{w}_j\|_2^2 + \sum_{i=1}^n \underbrace{\sum_{j \neq y_i} \max(0, \mathbf{w}_j^T \mathbf{x}_i - \mathbf{w}_{y_i}^T \mathbf{x}_i + m)}_{}$$

$$\frac{\partial}{\partial w_j} : \begin{cases} w_j^T \mathbf{x}_i - w_{y_i}^T \mathbf{x}_i + m \leq 0: & 0 \\ w_j^T \mathbf{x}_i - w_{y_i}^T \mathbf{x}_i + m > 0: & \mathbf{x}_i \end{cases}$$

$$\rightarrow 1(w_j^T \mathbf{x}_i - w_{y_i}^T \mathbf{x}_i + m > 0) \mathbf{x}_i$$

Computing The Gradient

$$\arg \min_{\mathbf{W}} \lambda \sum_{j=1}^K \|\mathbf{w}_j\|_2^2 + \sum_{i=1}^n \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T \mathbf{x}_i - \mathbf{w}_{y_i}^T \mathbf{x}_i + m)$$

$$\frac{\partial}{\partial w_{y_i}} : \sum_{j \neq y_i} 1(\mathbf{w}_j^T \mathbf{x}_i - \mathbf{w}_{y_i}^T \mathbf{x}_i + m > 0) (-\mathbf{x}_i)$$

Interpreting The Gradient

$$-\frac{\partial}{\partial w_j} : \underbrace{1(w_j^T x_i - w_{y_i}^T x_i + m > 0)}_{\text{blue bracket}} - x_i \underbrace{\}_{\text{red bracket}}$$

If we do not predict the correct class by at least a score difference of m ...

Want incorrect class's scoring vector to score that point lower.

Recall:
Before: $w^T x$;
After: $(w - \alpha x)^T x = w^T x - \alpha x^T x$